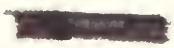
LOGICAL DESIGN OF A MICROPROGRAMMED SPECIAL-PURPOSE COMPUTER

ROBERT AUSTIN INGALLS

LOGICAL DESIGN OF A MICROPROGRAMMED SPECIAL-PURPOSE COMPUTER

by

Robert Austin Ingalls
Lieutenant, United States Coast Guard
B.S., United States Coast Guard Academy, 1960



Submitted in partial fulfillment
for the degree of

MASTER OF SCIENCE IN ENGINEERING ELECTRONICS

from the

UNITED STATES NAVAL POSTGRADUATE SCHOOL
December 1966

ABSTRACT

An investigation is made of the microprogramming approach to logical design of a digital machine. The digital processor designed performs general matrix manipulation and in particular is adapted to the requirements for implementing a Kalman-Weiner filter in a Track-While-Scan radar system.

A timing and data-flow analysis is made and micro-programs written to implement the required macros for the selected benchmark application. A detailed comparison is made of the operation of the micro-programmed processor to that of a current general purpose avionics type computer of similar main memory cycle time. Some conclusions are made concerning the optimizing of various parameters of a micro-programmed computer design.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF SYMBOLS

Symbol	Definition
A	A Register (16 bits)
A_L	The contents of the lower half, $(A_9 \ldots A_{16})$, of the A Register
a	The "a" field of a main instruction word containing a five bit opcode
b	The "b" field of a main instruction word containing the first eight bit operand address
C	Main Memory Address Register (8 bits)
c	The "c" field of a main instruction word containing the second eight bit operand address
d	The "d" field of a main instruction word containing the eight bit result address
F	Microinstruction Decoding Register (16 bits)
f	The "f" field of a microinstruction word containing the five bit opcode
H	Control Memory Address Register (8 bits)
I	I Counter Register (16 bits)
I_o	Initialization Register for I (16 bits)
i	The "i" field of a main instruction word containing a three bit index
J	J Counter Register (16 bits)
J_o	Initialization Register for J (16 bits)
j	The "j" field of a microinstruction word containing a three bit y field modifier
K	K Counter Register (16 bits)
K_o	Initialization Register for K (16 bits)
k	The "a" field of a microinstruction word containing eleven bits used with concurrent operation opcodes

Symbol	Definition
k_1 . . . k_{11}	The elements of the k field
M	Main Memory (256 x 16)
m	The column dimension of a matrix
N	Control Memory (256 x 16)
n	The row dimension of a matrix
Q	Q Register (16 bits)
R	Main Instruction Decoding Register (16 bits)
S	S Register (16 bits)
T	A cell in Control Memory used for temporary storage
W	A cell in Control Memory used for temporary storage
X	X Register (8 bits)
X_o	Initialization Register for X (16 bits)
Y	Y Register (8 bits)
Y_o	Initialization Register for Y (16 bits)
y	The "y" field of a microinstruction word containing an eight bit operand address
Z	Z Register (8 bits)
Z_o	Initialization Register for Z (16 bits)
(Y)	The contents of the Y Register
M⟨X⟩	The Main Memory cell addressed by the contents of the X Register
(N⟨Z⟩)	The contents of the Control Memory cell addressed by the contents of the Z Register
⟶	Indicates a transfer operation

Symbol	Definition
$(A_i \ldots A_j)$	The contents of the ith through the jth bit position of the A register. Register bit positions are numbered from left to right starting with one
p : q	Operation q is performed if p is true
58⟶A	The octal number 58 is put in the A Register
(A)⟶302	The contents of the A register are transferred to cell #302 in N

1. Introduction

The great majority of general purpose digital computers are designed around the concept of sequential performance of stored instructions. These instructions may be as simple as merely transferring a quantity from one place to another or as complicated as multiplying two quantities together. The process of performing one instruction is usually subdivided into two parts or cycles. The first cycle is called the "Read Next Instruction" (RNI) cycle and refers to the transfer of an instruction word from memory to a special instruction register. The second cycle is the "Execution" (EXEC) cycle and is concerned with the interpretation of the instruction word and execution of the specified operation.

Any computer operation can be reduced to a series of data transfers between various registers and/or memory locations. These transfers are controlled by opening and closing gates in the data transfer paths. Thus the execution of a specific instruction requires the generation of a specific sequence of transfer command signals. This process is represented in Figure 1 where the inputs are the code for the desired operation (opcode) and the address(es) of the operand(s) which are all contained in the instruction word. Of particular interest here is the method of implementing the "transfer command generator".

The Conventional Design Approach

The conventional manner of generating these signals is through the use of hardwired logic circuitry consisting of AND-OR-INVERT, NOR-NAND, or equivalent logic and FLIP-FLOPS. (See Figure 2). The instruction word is transferred to the instruction register where the opcode and address portions are decoded by separate decoding

matrices. A clock, which controls the timing of the basic computer, generates timing pulses which cause the timing and sequencing control to step through a series of states at precise time intervals. Each state generates a signal which is combined with the outputs of the decoding matrices in a control matrix. The control matrix in turn generates the proper gate control signals to execute each data transfer.

As the number of memory locations, registers, and opcodes increases, this circuitry rapidly becomes very complex and usually is a major portion of a computer's hardware. Another aspect of the usual computer design concept is that the computer designer determines the manner in which a given instruction is executed and this, being invariant once the computer is built, places a limit on the flexibility of the machine. Usually, to prevent this from being a serious limitation, the instruction set is carefully constructed to meet the anticipated needs of the user, but this of course, results in an even more complex "generator". One way of overcoming these disadvantages is by use of "Microprogramming".

The Microprogramming Approach to Computer Design

The microprogramming concept is not new, but until recently, it had not been utilized in commercially available, general-purpose computers.

Basically, microprogramming is another method of implementing the "transfer command generator". A definition which reflects the most characteristic feature of modern microprogramming design practice is as follows:

"Microprogramming extends the concept of a stored-program digital machine to include a second, lower or more basic level of stored in-

structions for purposes of either flexibility of function, economy of computer logical circuitry, or a combination of both".

This definition implies the use of a second memory containing stored-instructions each of which commands a particular data transfer. Thus by entering this second memory or control memory, as it is usually called, at the proper point and executing a given sequence of these second-level or microinstructions, the necessary series of transfer commands can be generated. The main opcodes are constructed to point to the proper entry point in control memory.

The general configuration for microprogramming is shown in Figure 3. The main opcode goes to the control memory address register and initiates the proper entry into the series of microinstructions stored in control memory. The microinstruction is transferred to a microinstruction register. From there, it is applied to a decoding matrix, along with the operand address and timing pulses. The output of this matrix is the necessary transfer command signals. Each microinstruction also specifies the next microinstruction of the desired sequence.

This, in general, is a brief outline and comparison of the conventional versus microprogramming approach to computer design. It is clear that the microprogramming idea hinges on the availability of a suitable efficient and economical control memory. The increasing capability of such small, fast "scratchpad" memories is evident in a number of recent reported designs.[4] The digital machine logic designer is presented with the potential for significant improvements in machine performance. It is the purpose of this thesis to explore and evaluate the microprogramming design approach. To this end the requirements of a typical real-time digital process are used to establish

the parameters of a machine. The design is then carried out, using principles of microprogramming. Programs are written for the specified application and the overall machine performance compared with that of a representative current military real-time control computer of similar arithmetic and main memory speed.

Block Diagram of
Transfer Command Generator

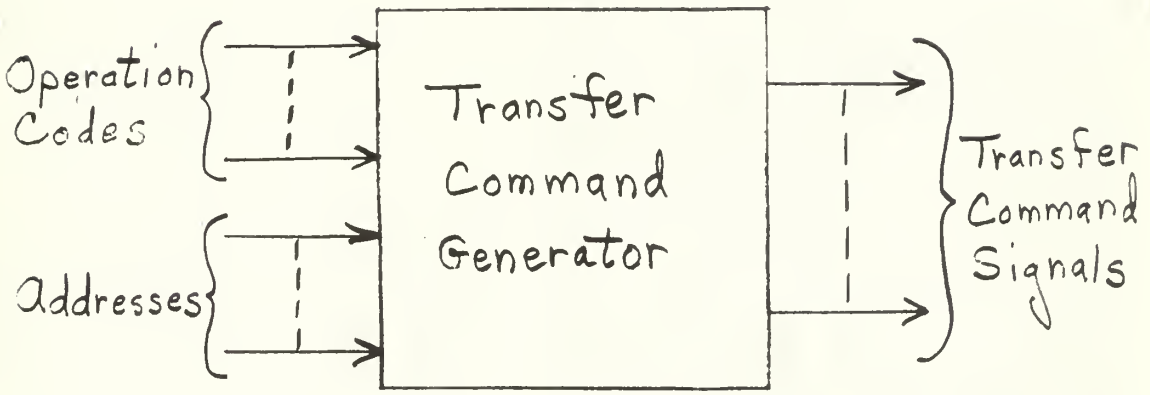


Figure 1.

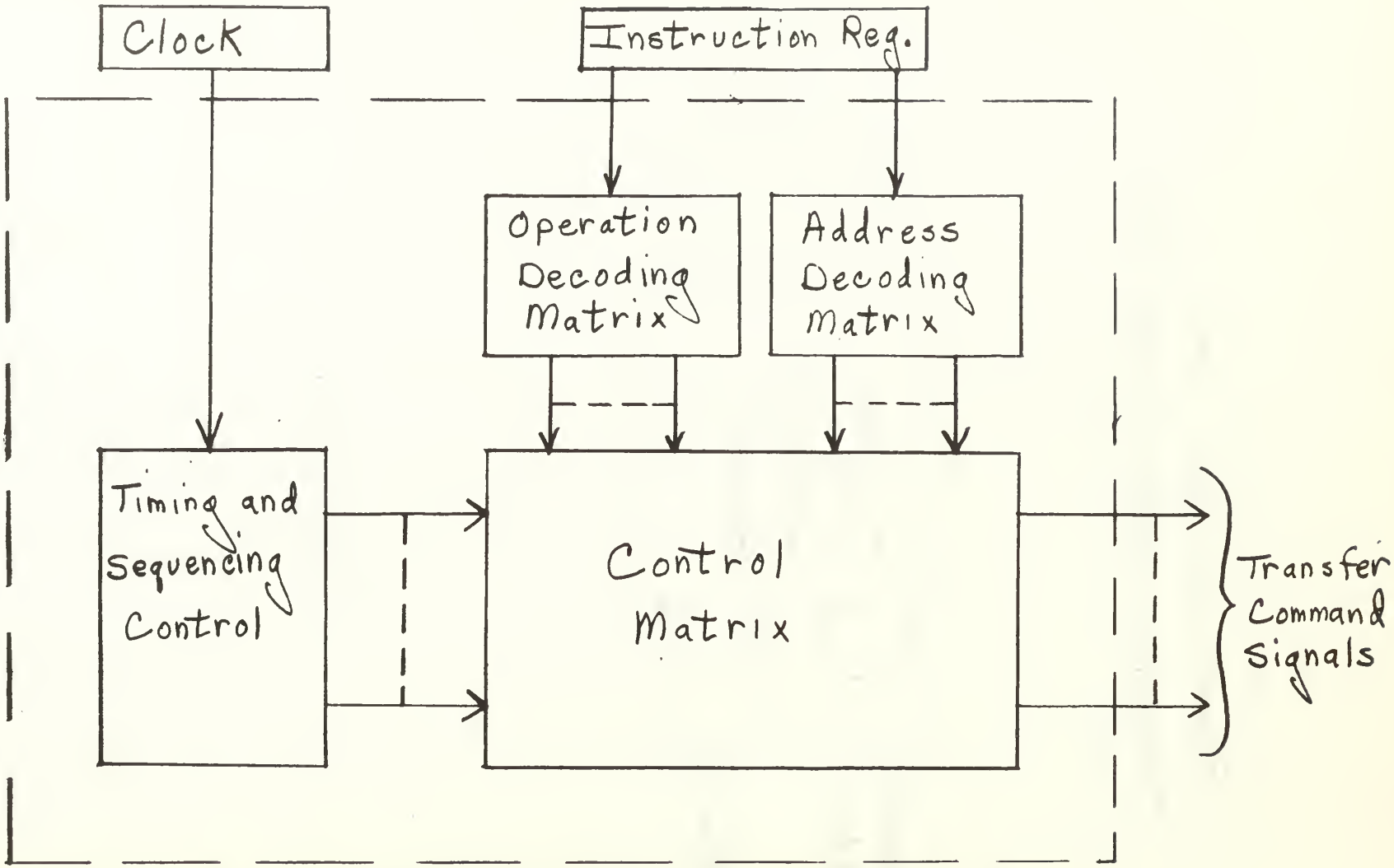


Figure 2. Conventional Design of TCG

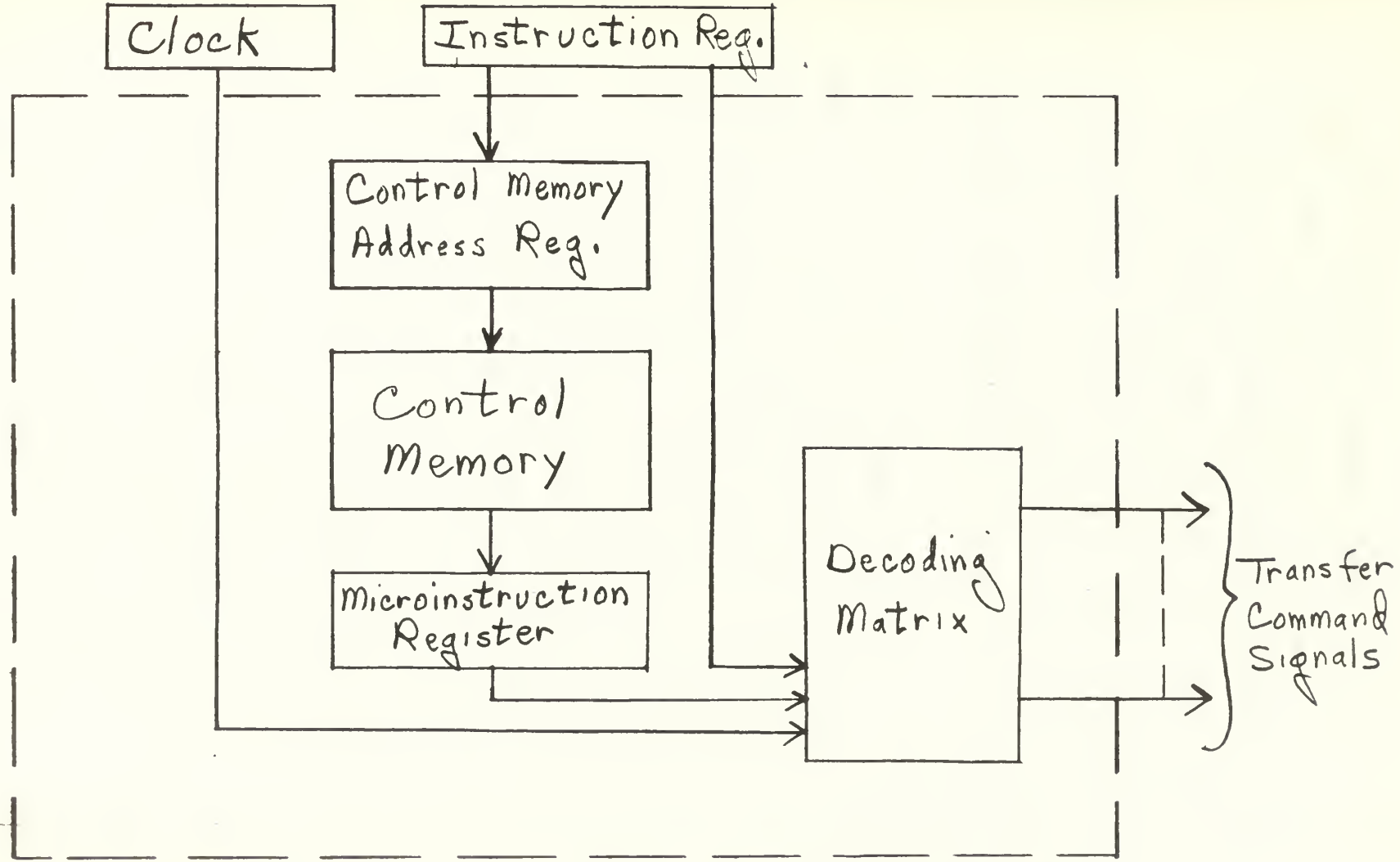


Figure 3. Microprogramming Design of TCG

2. Objective

In today's world of high speed aircraft and missiles, there is an ever increasing need for more complex control systems. These systems must be capable of observing and predicting many state variables and outputing accurately and quickly the many control signals required. The digital computer in conjunction with sampled-data control theory is helping to supply this need. Also with the advent of microelectronics these computers can be made small enough to make them feasible for aerospace applications. Thus there is a requirement for a computer to do a specialized job.

A common question that often arises at this point is one of "general purpose" computer versus "special purpose" computer. By "general purpose" is meant a computer capable of general scientific computation such as is usually found in industry, universities, and laboratories. "Special purpose" denotes a computer built to handle a specific type of problem and therefore is generally very limited as to general-purpose capabilities. The intuitive answer would be to choose a "special-purpose" computer for a specialized job. However, this may not be the right answer.

A general-purpose computer can usually be programmed to perform the function of any special-purpose computer and therefore is adaptable to a variety of special-purpose applications. Also the GP computer is more likely to be commercially available, thus saving the user the R & D costs associated with producing a new machine of limited use.

On the other hand, the special-purpose idea is not without its advantages. Since it must perform only specified tasks, it can be designed to do them with maximum efficiency, and the hardware can be

minimized which will give the SP computer a definite advantage with regard to size, weight, and reliability. It is thought that the microprogramming approach to computer design could produce a flexible computer in a small enough package to be a useful compromise solution and it is the objective of this thesis to investigate this hypothesis.

In order to make a quantitative analysis, it is necessary to assume a specific situation, assign realistic parameters, and evaluate the results. The application of a computer as a digital filter in a Track-While-Scan airborne radar system was chosen as the specific application. In this case, the range to a target is sampled periodically in the presence of noise. It is desired to filter these range measurements or samples to produce a more accurate estimate of the target's actual range at any given sampling instant. R.E. Kalman developed a set of recursive equations based on sampled-data theory, commonly known as the "Kalman Filter", for this purpose. A block diagram of this filter is shown in Figure 4, where the single lines indicate scalar quantities, and the double lines indicate vector quantities. The actual equations are shown in detail in Appendix VII. A fourth-order system was assumed with only one state variable observable. Therefore the filter input is a scalar whereas the output is a vector consisting of the predicted values of all four state variables. It will be further noted that the filter equations are formulated in matrix notation and it is this fact that led to the design of a computer which specialized in matrix arithmetic.

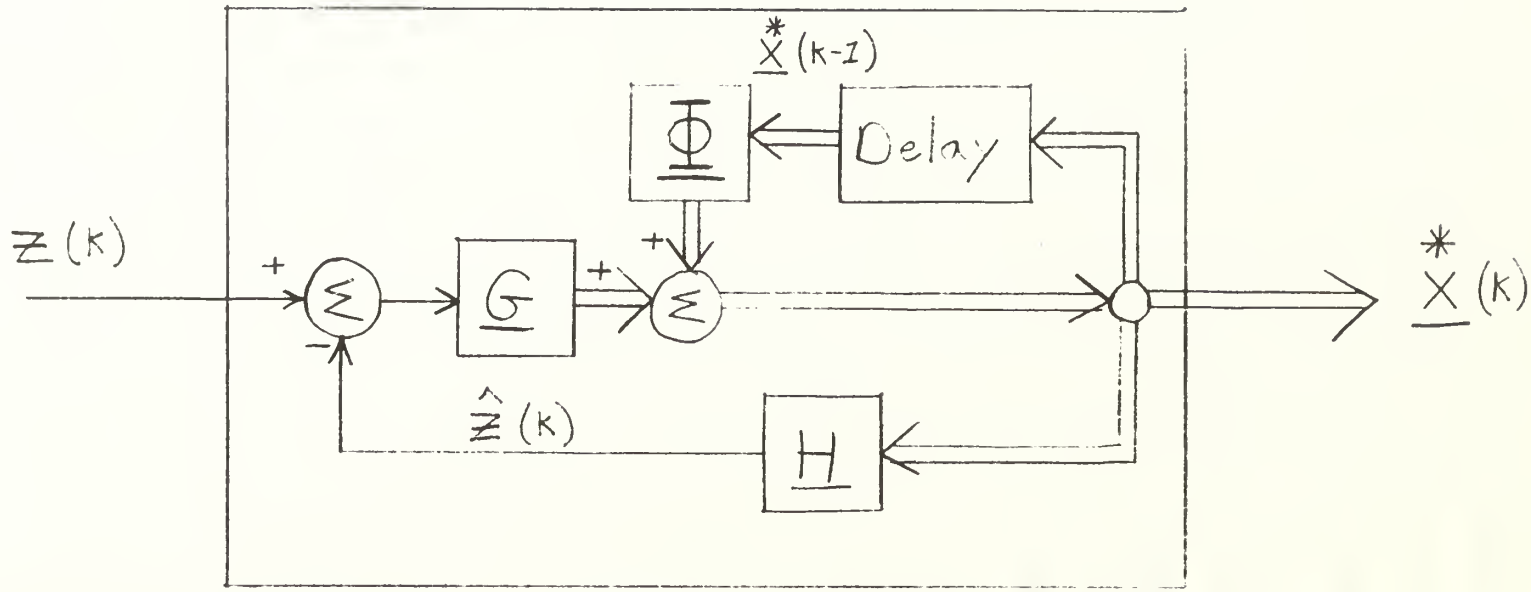


Figure 4. Block Diagram of Kalman Filter

3. Computer Design

Hardware

It is appropriate to begin by considering the basic word structure, since many computer parameters are directly related to this characteristic. A computer word, composed of a sequence of binary digits (bits), generally is interpreted in one of two ways. Either the entire word represents data (a number) in binary or it represents a computer instruction. In the latter case the word is generally divided into at least two parts, one being the code indicating the operation to be performed (opcode), and the other giving the address in memory of the operand or the operand itself. Thus the first two decisions required are 1) how long should the word be, and 2) how to subdivide the word when using it as an instruction. In this case these decisions had to be arbitrary, but an attempt was made to make them realistic.

It was assumed that this computer would be used as a digital filter and predictor in a Track-While-Scan radar system and that the maximum data quantity would be 1000 miles of range to an accuracy of .1 miles. To represent this quantity in binary would require 14 bits. Allowing one bit for addition overflow and one bit for sign, a total word length of 16 bits was chosen.

The second decision regarding how to subdivide the word is more complicated due to its interaction with other facets of the computer structure. It is generally desirable to allow any memory location to contain an operand. Therefore the number of bits alloted for operand address immediately determines the size of the memory or vice versa, using the relation that n bits can specify 2^n locations. In a similar fashion, the number of bits alloted to opcodes immediately prescribes

the maximum possible number of operations available. Another side to this problem peculiar to the microprogramming concept is the need for compatibility with the separate control memory which contains the microprograms. Despite the various dependencies on this decision mentioned above, it was actually a software consideration which really influenced the final choice.

It seemed desirable to be able to specify a complete operation in one instruction. In other words, an instruction would give the opcode, the locations of the two operands (in the more general case), and the location of the result. To do this it was decided to subdivide the word into 2 equal parts of 8 bits each and actually use 2 successive words for each instruction. The upper portion (first half) of the first word would contain the opcode, and the lower portion would contain one operand location. The second word would contain the second operand location, if any, and the location of the result in the upper and lower sections respectively. An immediate consequence of this decision is that the main memory is limited to 256 words which may be too small, but it was decided to evaluate this at the conclusion of the project.

At this point it is necessary to look at some of the special features required to implement the microprogramming concept. The most important one is the Control Memory (designated N) which contains the microprograms. This memory is separate from the Main Memory (designated M) and is usually an order of magnitude faster than M. The reason for this latter characteristic will be explained later. The microprograms appear much like subroutines. They are in effect "called" by the opcodes in the main program and are given the operand and result

locations as arguments. The microprograms in turn consist of micro-instructions which are executed sequentially. They must be written to completely perform the desired operation including the considerable bookkeeping inherent in matrix arithmetic.

Of course the question of how big to make the Control Memory has to be decided and here again 256 words were chosen. This means that an 8 bit address is required as was true for the Main Memory. This latter fact was not a coincidence by any means, but was done deliberately for a reason which will be explained.

The next step in the design process was to look in detail at the type of arithmetic operations to be performed. It was decided to consider the following five matrix manipulations:

1) Matrix Addition (MATAD)

2) Matrix Subtraction (MATSB)

3) Matrix Transposition (MATPS)

4) Matrix Multiplication (MATML)

5) Matrix Inversion (MATIN)

Matrix Multiplication was selected in particular as one of the more complicated yet typical matrix operations. It is necessary to store the elements of a matrix in memory in a specified sequence. This author chose to store them by rows reading the elements from left to right in conventional matrix notation. In the two cells immediately preceding the elements, the number of rows, n, and the number of columns, m, of the matrix would be stored in that order. Thus the matrix A would be stored in memory from location L onward, as follows:

Memory Location	Contents of cell
L	n
L + 1	m
L + 2	A_{11}
L + 3	A_{12}
.	.
.	.
.	.
L + m	A_{1m}
L + m + 1	A_{21}
.	.
.	.
.	.
L + (m x n) + 1	A_{nm}

Matrix Multiplication (A x B = C), is performed by means of the following algorithm.

$$C_{ij} = \sum_{k=1}^{m_A=n_B} A_{ik}B_{kj} \qquad \begin{matrix} i = 1, 2, \ldots, n_A \\ j = 1, 2, \ldots, m_B \end{matrix}$$

It was noted that at any given step in the above computation it is often the case that different elements of three matrices are referenced. Therefore to facilitate the acquisition of these elements as needed, it was decided to have three address registers, one for each matrix. Further, these registers would be able to address any location in the Main Memory or the Control Memory; hence the compatibility in size between the two memories. Also the three address registers (designated X, Y, and Z) would have the capability of being incremented or decremented in one step to facilitate reiterative steps being performed on successive matrix elements.

One final decision required to complete the specification of the Control Memory is the word length. Obviously this would have to be something greater than eight bits since the Control Memory contains

256 words. Although there is no definite reason, sixteen bits was chosen to permit temporary storage of little used microprograms in the Main Memory if the Control Memory proved to be too small.

Referring again to the matrix multiplication process, it was found that not only was it necessary to reference the elements of a matrix sequentially, but often this process had to be repeated more than once. To facilitate the reinitialization of an address register, three cells in the Control Memory (designated X_o, Y_o, and Z_o) were set aside to hold the initial address of each address register and have the built-in capability of transferring this quantity to their respective address registers in one step. Of course, some type of counter is always essential in this kind of operation, so three cells in the Control Memory (designated I, J, and K) were set aside as counters with a one-step increment or decrement capability. The reinitialization problem occurs here also; so I_o, J_o and K_o were designated with the one-step transfer capability similar to X_o, Y_o, and Z_o. A sixteen bit arithmetic register (A register) was necessary for basic addition and subtraction and to provide a built-in multiply and divide capability, two other sixteen bit registers (Q and S) were required. As is the common practice, the A and Q registers could be connected together to form a double length, AQ, register with a long right or long left shift capability for use in the multiplication and division process.

Finally two eight bit registers, C and H, to act as basic memory address registers for M and N respectively were provided, along with a sixteen bit F register which is used to decode the microinstructions. This completed the basic hardware characteristics of the computer and the resulting block diagram with data transfer paths is shown in Appen-

dix I. The single-ended arrows indicate unidirectional data flow, while the double-ended arrows indicate a bidirectional data flow, not necessarily over the same wires, however.

Instruction Format and Operation Codes

The eight bits available for operation codes in a main program instruction word were subdivided into two parts. The first part consists of five bits and actually addresses the particular cell, in the first thirty-two cells of Control Memory, which acts as the entry point for the particular microprogram required. The second group of three bits acts as an index group. It was found in studying the microprogram requirements of MATAD and MATSB that these two operations required identical "bookkeeping" and only the actual arithmetic operation was different. Therefore the same microprograms could be used provided there was a microinstruction within the microprogram to sense the appropriate arithmetic operation. This microinstruction references the three bit index of the main instruction to make this decision. To the programmer, MATAD and MATSB would each have a specific eight bit opcode, but actually only the last three bits would be different. Thus, in general, the use of the index permits minor branching within a given microprogram. Only the opcodes for the five basic matrix operations were established and they are listed in the Appendix II. Obviously there is a capability for a great many more.

There were several possibilities considered regarding the format of a microinstruction and the sequencing of these instructions. One, in particular, was to incorporate into an instruction the address of the next instruction to be performed, thus minimizing the number of times a particular instruction appeared in the Control Memory. This

idea appeared to require too complex a coding scheme to achieve the desired flexibility, but an evaluation of the idea will be discussed in the Conclusions Section. The system actually implemented was the basic two cycles used in most computers today. The first cycle is usually referred to as the "Read Next Instruction" (RNI) cycle and does exactly what the name implies. It transfers the next instruction from memory to a special decoding register. In this particular instance this would be a transfer from N to F. The second cycle is called the "Execution" (EXEC) cycle and includes the decoding of the instruction word to determine the required operation and the location of the operand, if any, as well as the performance of the actual operation. A particular characteristic of this mode of operation is the sequential execution of instructions stored in memory. This may well have been expected from the use of the term "Read Next Instruction". A further implication here is the need for jump or branch type microinstructions.

After a certain amount of trial and error, the microinstruction word was divided into three parts. The first part consists of five bits which specify a basic micro-opcode. The next three bits are used as an opcode modifier. The last part consists of eight bits which may specify the operand address or an eight bit operand itself.

To increase the overall speed of the computer, the idea of permitting concurrent operations was considered. By studying the basic matrix operations, it was observed that quite frequently more than one of the address registers required incrementing between successive data transfer or arithmetic steps. A similar requirement was found to exist for the counters. Therefore a variation of the microinstruction format was developed using each bit in the last two portions of the word to specify concurrent operations. To illustrate this idea, an example

using the "add One Direct" (AOD) microinstruction is shown.

Example:

AOD 0 0 0 0 0 1 0 0 0 1 1
(not used) I J K A X Y Z

This microinstruction will add one to the contents of the J, X, and Z registers simultaneously.

A companion microinstruction, "Subtract One Direct" (SOD), was also established.

A third concurrent operation which appeared useful allows the simultaneous transfer of the contents of the A register to several different locations. This permits the referencing of this quantity at later points in the microprogram. The microinstruction is "Store A" (STA) and has a similar format to the AOD instruction.

The remaining opcodes which were established are of a straightforward nature and appear in Appendix III with detailed instructions for their use.

Microprogram Structure

This third and final part of the Computer Design concerns the actual structure and timing of the complete microprogram necessary to perform a main instruction. Looking at the main program, it appears that each main instruction is performed using the two-cycle system of RNI and EXEC described earlier. Actually this requires two separate microprograms. The first microprogram performs the RNI cycle. It transfers the first word[1] of the main instruction to the R register, further transfers the two address portions to specific locations, and then repeats the whole operation for the second word of the main instruction.

[1]Each main instruction uses two consecutive words in the main memory.

Actually the same RNI microprogram is used for every instruction which means that a given part of every main instruction is always transferred to the same location during the RNI cycle. A detailed description of the RNI microprogram is shown in Appendix IV in a symbolic form using the notation found in [1].

The EXEC cycle is performed by a separate microprogram which of course depends on the operation required. The main opcode actually "calls" the appropriate microprograms as mentioned earlier.

Looking now at the microprogram structure, it is found that each microprogram consists of a sequence of microinstructions and in general appears very similar to programs used in modern general-purpose computers. Here again the basic two cycle system is used but in this case the cycles are performed by means of hardwired logic as in the conventional computer. The actual writing of the microprograms is exactly the same as writing a conventional subroutine. The entering arguments which in this case are the operand addresses, are in known locations and the desired location of the result has also been specified.

To illustrate some of the concepts and procedures that have been developed in this section, an example is presented in Appendix IV. The main opcode calls for a transfer of a block of data from the Main Memory to the Control Memory. The starting address of the block in M, the number of words in the block and the starting address of the block in N are also specified in the main instruction word. The microprogram RNI cycle is described and is understood to occur between every microinstruction.

4. Computer Test

Now that a microprogrammed computer has been designed, it is necessary to apply a test to evaluate the results. Of course, one could determine the absolute times required for various operations, but this would not indicate the true worth of the concept. A more meaningful procedure would be to compare the microprogrammed computer against one of conventional design, intended for similar usage.

The computer chosen for this comparison was the UNIVAC 1830 Avionics computer. This computer is a miniaturized, solid state, general-purpose computer, advertised for use in airborne and missile command and control systems. This of course, is the type of usage envisioned for the microprogrammed computer and is the chief reason for choosing the 1830 for comparison tests. The 1830 is an outgrowth from the NTDS Unit Computer, AN/USQ-20, which was also built by UNIVAC and it uses the same instruction set. A complete technical description and listing of the instructions can be found in [3]. The basic characteristics and instructions have been excerpted from [3] and are presented in Appendix VI.

It is common practice when comparing one computer versus another to prescribe certain "benchmark" programs. These "benchmark" programs are special tasks which occur frequently in computer usage and require more than the simplest programming to implement. They usually involve a certain amount of "bookkeeping" and arithmetic computation. In general, they are chosen to exercise as many of the hardware and software characteristics of a computer as possible. One of the most frequently used "benchmark" programs is matrix multiplication. This not only checks the efficiency of the multiplication routine, but also shows

how well a computer can do multiple addressing and loop indexing.

Each "benchmark" program is programmed in the language of the computers to be compared, and then the running times are either calculated or measured. The resulting times are usually a good basis of comparison. Other factors, such as ease of programming or flexibility of instructions, may also be important depending on the user's application.

In this particular case, where the computers are intended as elements of a control system, matrix arithmetic appeared to be a source of appropriate "benchmark" programs, since many of the control laws are formulated as recursive matrix equations.

As a preliminary test, the five basic matrix operations (addition, subtraction, transposition, multiplication, and inversion) were programmed for both computers. These programs will handle matrices up to the dimensions of 7 x 7 and are listed in detail in Appendix V.

The major "benchmark" program chosen for the test was the Kalman Filter. The filter is a set of three recursive matrix equations, which predict the state variables of a system based on noisy measurements of one or more of the variables. The particular feature of this filter is its ability to reduce the variance of the predictions due to a noisy environment. A detailed description of the filter can be found in [2]. Appendix VII lists the equations, explains the notation used and the equivalent notation that appears in the programming, and gives the matrix dimensions assumed for the calculation of running time.

The relative timing required for both the preliminary programs and the major "benchmark" test is presented in the Evaluation and Conclusions section.

5. Evaluation and Conclusions

The basis for evaluating the microprogrammed computer was a comparison of the execution times required by the microprogrammed computer and the 1830 computer for each test program. For each microprogram, a formula was derived which gives the execution time in terms of minor cycles as a function of the operand matrix dimensions. Similar formulas were derived for each subroutine with the results in terms of main memory cycles. These formulas are shown in Table I. The variables, n and m, are subscripted in the formula for MATML since this is the only program that can have two operands of different dimensions.

The first or preliminary test was to compare the execution times for each of the five basic matrix operations. All operand matrices were assumed to be 4 x 4 and using the formulas mentioned above, these times were calculated and are also shown in Table I.

The final and major test was to compare the execution times for the "benchmark" program, the Kalman Filter. Using the matrix dimensions shown in Appendix VIII, the time to complete one iteration of the filter equations was computed for each computer and the results appear as the final entry in Table I.

Up to this point no absolute time scale had been assigned to the microprogrammed computer. However, to make the comparison of the test results more meaningful, it was necessary to do so. Since the advertised memory cycle time for the 1830 computer was 4 microseconds, it seemed only fair to assign that same value to the microprogrammed computer's main memory or major cycle. This then sets the minor cycle at .4 microsecond, based on the earlier assumption that the control memory was to be an order of magnitude faster than the main memory.

These timing assumptions were used to convert the test results in cycles to absolute values of time, which appear as the other two data columns in Table I.

The first observation one might make is that the microprogrammed computer appears to be 3-5 times faster than the 1830 computer. This was the anticipated result but the magnitude of the difference should not be considered as a general figure. It can be noted by looking at the subroutines written for the 1830 computer, that a considerable number of instructions were used to transfer the arguments into the subroutine. This was necessary since the 1830 has no provisions for indirect addressing. As a result the subroutine execution times are somewhat longer than might be expected on a machine having this feature. The percentage of the total execution time consumed by this task will decrease however, as the matrix dimensions increase, since the argument location is performed only once, whereas the computational instructions are repeated depending on the matrix dimensions.

A second meaningful observation, though perhaps not apparent, is a comparison of the number of cycles required for each program. These figures as shown are indicative of the number of instructions or steps required by each computer to perform the given matrix operation. It will be noted that the microprogrammed computer requires approximately 3 times the steps or instructions to accomplish the same task as the 1830. This fact bears out the contention of this author that effective utilization of the microprogramming concept requires the use of a high-speed control memory. Quite obviously, the microprogrammed computer would have been approximately 3 times slower than the 1830 if the microprograms had been stored in the main memory, or an

auxiliary memory of the same speed.

Several conclusions are apparent at this point, one of which is that the memory sizes should be increased. The control memory was not large enough as originally designed to contain the five matrix microprograms, much less allow for temporary storage areas. Also the main memory size of 256 appears small as was mentioned earlier. It is felt that making the main memory 4096 words and the control memory 1024 words would be adequate. Using these two figures, several other changes are also recommended.

The main memory word should be increased to 20 bits. With 12 bits required to address 4096 words, 8 bits remain for opcodes. This 8 bit section could be further divided to allow 6 bits to specify 64 entry points in the control memory and 2 bits as the opcode modifier. The ability to transfer matrix elements into the control memory, when performing an operation requiring multiple accessing of a given element still seems desirable, and thus the control memory word should be at least 20 bits in this case.

Actually there is a trend today by those manufacturers utilizing microprogramming in some form to use a long control word. The advantage of this is to make possible a greater combination of concurrent operations specified by one microinstruction. Looking back at the microprograms, one can see that there are many sequential operations performed which are independent of each other, and thus could occur simultaneously if the appropriate instruction could be formulated.

If a 20 bit control memory were used, 10 bits of course would be required to address any location in the control memory. The remaining 10 bits could be subdivided to give 6 bits of opcode specification and

4 bits as an index or indices. This would allow 64 opcodes which should be adequate to designate all the necessary operations for general programming as well as those needed for I/O and special microsubroutines such as multiply and divide.

The use of the dual-purpose address registers, X, Y, and Z, and the counters, I, J, and K, along with their initialization registers was very convenient and should be retained. The X, Y, and Z registers would have to be increased to 12 bits in this proposed version.

Another question, regarding the advantages of microprogramming, concerns the amount of logic circuitry required. Generally speaking, the microprogramming concept is supposed to essentially substitute software for hardware. Of course, eliminating the operation counter would cause this reduction, but a certain amount of logic is necessary to implement the specified microinstructions, particularly those calling for various combinations of concurrent operations.

Whether there would be a net reduction of logic in the microprogrammed computer as designed cannot be answered definitely without first doing a complete circuit design.

One final consideration is that of size and weight. The advertised figures for the 1830 are 1.1 cubic feet and 50 pounds. It is felt that the microprogrammed computer should be no larger than this, and with the present advances in integrated circuits, would probably be even smaller in physical size.

In conclusion, it is felt that implementation of the microprogramming concept of computer design can provide a user with a special-purpose computer capable of being reprogrammed to fit a variety of applications at a competitive cost.

TABLE I

Microprogram Execution Time Formulas

MATAD

$$112 + 43nm$$

MATSB

$$112 + 41nm$$

MATPS

$$128 + 40mn + 4m - 12n$$

MATML

$$229 + n_A \left[17n_B + 4 + m_B(80n_B + 19)\right] + 3n_B \left[14m_B - 4\right] + 4m_B$$

MATIN

$$120n^3 + 116n^2 + 21n + 62$$

Subroutine Execution Time Formulas

MATAD

$$78 + 11nm$$

MATSB

$$78 + 11nm$$

MATPS

$$20nm + 13n + 63$$

MATML

$$75 + n_A \left[18 + m_B(31n_B + 25)\right]$$

MATIN

$$39n^3 + 91n^2 + 12n + 104$$

Computer Test Results

Operation	Microprogrammed Computer		UNIVAC 1830 Computer	
	minor cycles	microseconds	major cycles	microseconds
MATAD	688	275	254	1016
MATSB	656	262	254	1016
MATPS	736	294	435	1740
MATML	8981	3592	2531	10124
MATIN	14646	5858	4104	16416
Kalman Filter Program	40977	16391	13142	52568

BIBLIOGRAPHY

1. Chu, Y. Digital Computer Design Fundamentals. McGraw-Hill Book Company, Inc., 1962

2. Hallas, H. G. B. The Hybrid Simulation of a Tactical Weapon Discrete Filter-Controller, U.S. Naval Postgraduate School Master's Thesis, 1966

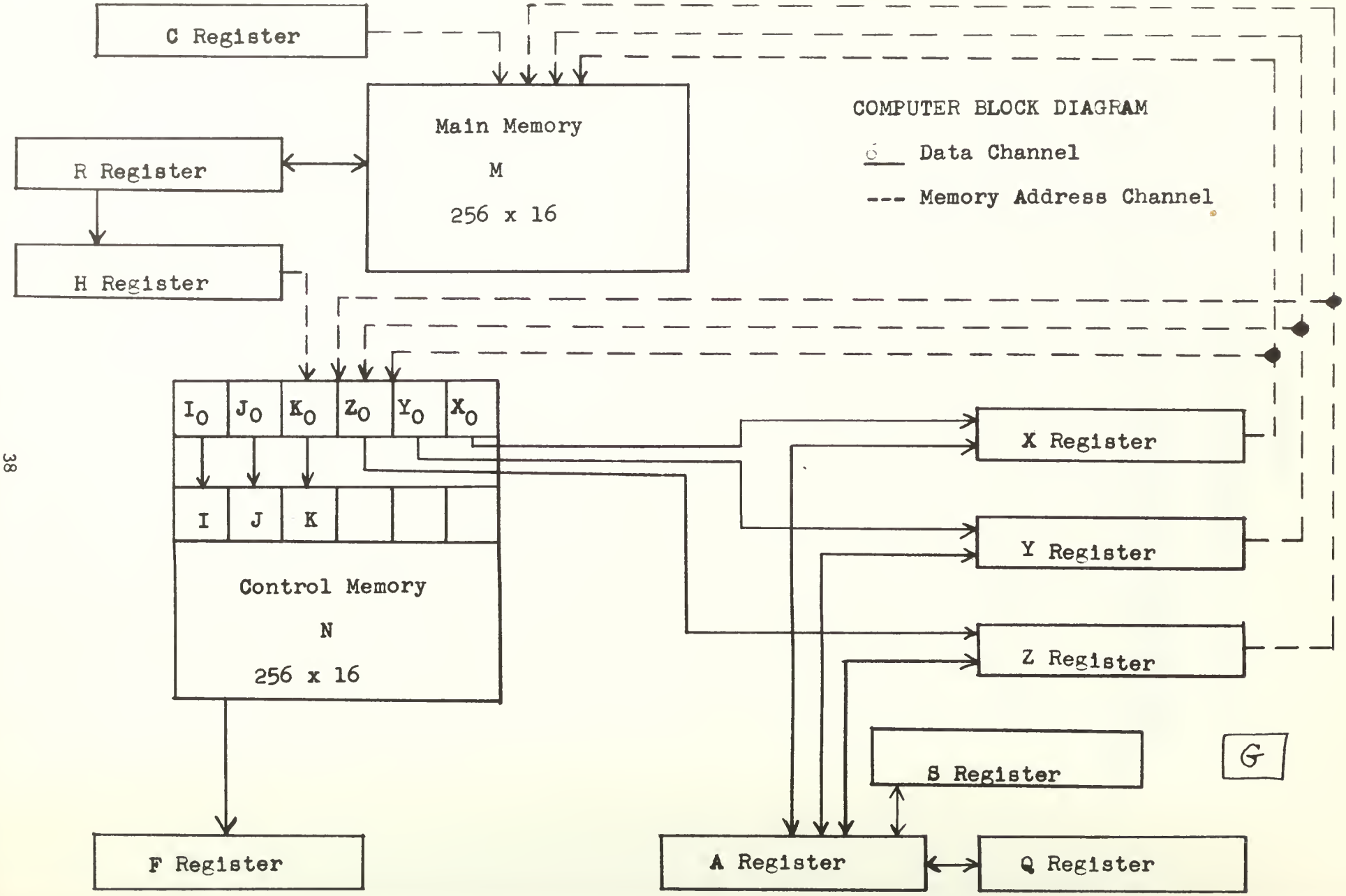
3. Technical Description, UNIVAC CP-823/U Military Computer.

4. A High Speed Integrated Circuit Scratchpad Memory. I. Catt, E. C. Gorth, D. E. Murray, Proc. of Fall Joint Computer Conference 1966. AFIPS vol. 29.

APPENDIX I

COMPUTER BLOCK DIAGRAM

This Appendix contains a block diagram showing the physical organization of the computer and the data and address channels.

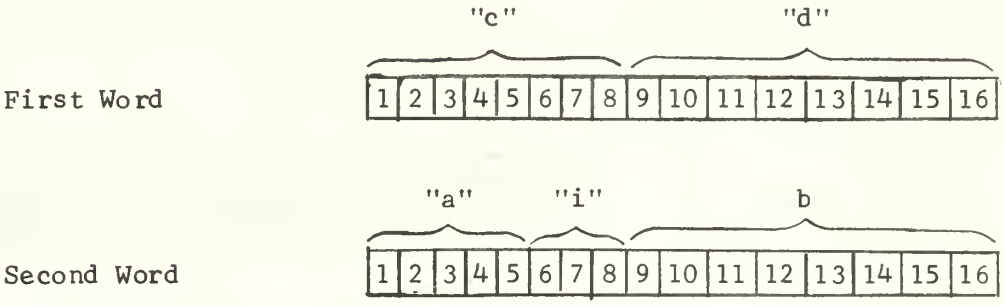

C Register
Main Memory
M
256 x 16
COMPUTER BLOCK DIAGRAM
Data Channel
Memory Address Channel
R Register
H Register
I_0 J_0 K_0 Z_0 Y_0 X_0
I J K
Control Memory
N
256 x 16
X Register
Y Register
Z Register
S Register
G
F Register
A Register
Q Register

APPENDIX II

MAIN INSTRUCTION FORMAT

This Appendix contains a description of the main instruction format and a listing of the main instruction opcodes.

A main instruction actually consists of two sixteen-bit computer words. The division of these two words into fields is shown below.



The "a" and "i" fields contain the operation code. These codes are described below. The "b" field contains the address of the first operand and the "c" field contains the address of the second operand, if any. Finally the "d" field contains the address at which the result is to be stored.

Although each main opcode appears as an eight bit quantity, it actually consists of an "a" and an "i" field. The five bit "a" field specifies the microprogram entry point and the "i" field is used as a branching indicator within a microprogram.

The following is a list of the main opcodes used in this project.

Opcode	Field	
	"a"	"i"
MATAD	00101	001
MATSB	00101	010
MATPS	00110	001
MATML	00111	010
MATIN	01000	000

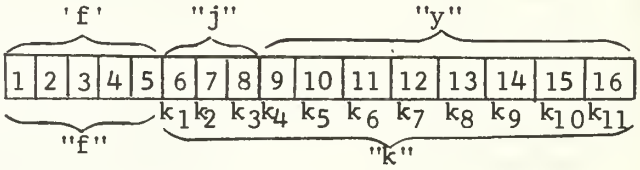
APPENDIX III

MICROINSTRUCTION FORMAT

This Appendix contains a description of the microinstruction format and a listing of the microinstruction opcodes.

Shown below are the two possible formats for a microinstruction word. The "f", "j", "y" configuration is used where the instruction entails a single operation. The "f", "k" configuration is used with instructions calling for concurrent operations.

Microinstruction Word



The "k" field is actually composed of eleven single bit fields labelled k_1 through k_{11}.

The micro-opcodes used in this project appear on the following pages with the details for their use. The actual bit patterns are shown in octal to clarify the presentation.

LDA j y 01 Load A

This instruction transfers the sixteen bit quantity found in the location specified by j and y to the A register.

j = 1 and	y = 1:	Load (M<x>)
	y = 2:	Load (M<Y>)
	y = 3:	Load (M<Z>)
	y = 5:	Load (N<X>)
	y = 6:	Load (N<Y>)
	y = 7:	Load (N<Z>)
	y = 11:	Load (X)
	y = 12:	Load (Y)

y = 13:	Load (Z)
y = 15:	Load (S)
y = 16:	Load (Q)
j = 2:	Load the contents of cell in N whose address is given in y
j = 3:	Load the quantity in y

STN j y 02 Store N

This instruction transfers the sixteen bit quantity in the A register to the cell in N addressed by y. The j field is not used in this instruction.

SAL j y 14 Store A Lower

This instruction transfers the contents of the lower half of the A register to the lower half of the cell in N specified by y. The j field is not used.

ADD j y 03 Add to A

This instruction adds to the A register the quantity in the location specified by j and y. The codes for the j and y fields are the same as for the LDA instruction.

SUB j y 04 Subtract from A

This instruction subtracts from the A register the quantity in the location specified by j and y. The codes for the j and y fields are the same as for the LDA instruction.

JPU j y 12 Unconditional Jump

This instruction causes a unconditional jump to the microinstruction in the cell addressed by y. The j field is not used.

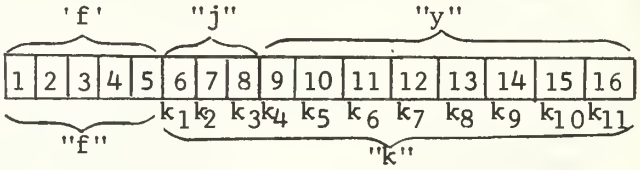
APPENDIX III

MICROINSTRUCTION FORMAT

This Appendix contains a description of the microinstruction format and a listing of the microinstruction opcodes.

Shown below are the two possible formats for a microinstruction word. The "f", "j", "y" configuration is used where the instruction entails a single operation. The "f", "k" configuration is used with instructions calling for concurrent operations.

Microinstruction Word



The "k" field is actually composed of eleven single bit fields labelled k_1 through k_{11}.

The micro-opcodes used in this project appear on the following pages with the details for their use. The actual bit patterns are shown in octal to clarify the presentation.

LDA j y 01 Load A

This instruction transfers the sixteen bit quantity found in the location specified by j and y to the A register.

j = 1 and	y = 1:	Load (M<x>)
	y = 2:	Load (M<Y>)
	y = 3:	Load (M<Z>)
	y = 5:	Load (N<X>)
	y = 6:	Load (N<Y>)
	y = 7:	Load (N<Z>)
	y = 11:	Load (X)
	y = 12:	Load (Y)

y = 13:	Load (Z)
y = 15:	Load (S)
y = 16:	Load (Q)
j = 2:	Load the contents of cell in N whose address is given in y
j = 3:	Load the quantity in y

STN j y 02 Store N

This instruction transfers the sixteen bit quantity in the A register to the cell in N addressed by y. The j field is not used in this instruction.

SAL j y 14 Store A Lower

This instruction transfers the contents of the lower half of the A register to the lower half of the cell in N specified by y. The j field is not used.

ADD j y 03 Add to A

This instruction adds to the A register the quantity in the location specified by j and y. The codes for the j and y fields are the same as for the LDA instruction.

SUB j y 04 Subtract from A

This instruction subtracts from the A register the quantity in the location specified by j and y. The codes for the j and y fields are the same as for the LDA instruction.

JPU j y 12 Unconditional Jump

This instruction causes a unconditional jump to the microinstruction in the cell addressed by y. The j field is not used.

ZJP j y 11 Zero Jump

This instruction causes a jump to the microinstruction in the cell addressed by y if the counter specified by j is zero. Otherwise the next microinstruction is executed.

j = 1:	Sense K, jump if zero
j = 2:	Sense J, jump if zero
j = 4:	Sense I, jump if zero

NJP j y 13 Non-Zero Jump

This instruction causes a jump to the instruction in the cell addressed by y if the counter specified by j is non-zero. Otherwise the next instruction is executed.

j = 1:	Sense K, jump if non-zero
j = 2:	Sense J, jump if non-zero
j = 4:	Sense I, jump if non-zero

AJP j y 15 A Jump

This instruction causes a jump to the instruction addressed by y if the conditions of j are satisfied. Otherwise, the next instruction is executed.

j = 1:	Jump if A = 0
j = 2:	Jump if A $\neq$ 0

JPI j y 10 Index Jump

This instruction causes a jump to the instruction in the cell addressed by y if the contents of j field is identical to the i field of the main instruction contained in the R register. Otherwise the

next instruction is executed.

The next group of micro-opcodes are those which specify concurrent operations. Each element of the k field specifies a particular operation and by setting any combination of the k field elements to 1, any desired group of concurrent operations can be specified.

STA k 10 Store A

This instruction transfers the sixteen bit quantity in the A register to the locations indicated by the k field elements which are set to 1.

$k_1 = 1$: Store A in M<X>
$k_2 = 1$: Store A in M<Y>
$k_3 = 1$: Store A in M<Z>
} only one element may be set at a time

$k_4 = 1$: Store A in N<X>
$k_5 = 1$: Store A in N<Y>
$k_6 = 1$: Store A in N<Z>
} only one element may be set at a time

$k_7 = 1$: Store A in X

$k_8 = 1$: Store A in Y

$k_9 = 1$: Store A in Z

$k_{10} = 1$: Store A in S

$k_{11} = 1$: Store A in Q

Note that only one element of k_1, k_2, and k_3 may be set to 1 at a time. This is also true for the k_4, k_5, and k_6 group.

AOD k 05 Add One Direct

This instruction increments by one the contents of the register indicated by the elements of the k field which are set to 1.

k_1, k_2, k_3, k_4 — not used

k_5 = 1: Increment I

k_6 = 1: Increment J

k_7 = 1: Increment K

k_8 = 1: Increment A

k_9 = 1: Increment X

k_{10} = 1: Increment Y

k_{11} = 1: Increment Z

SOD k 06 Subtract One Direct

This instruction decrements by one the contents of the registers indicated by the elements of the k field which are set to 1. The k field element codes are the same as for the AOD instruction.

RXF k 07 Right Transfer

This instruction transfers the quantity in an initialization register to its companion counter or address register. Any combination of the six possible transfers may be specified by setting the appropriate k elements to 1.

Example

k_6 = 1 and k_{11} = 1

This means transfer the contents of Y_o to Y and the contents of K_o to K simultaneously.

k_1, k_2, k_3: not used

$k_4 = 0$: (always zero)

$k_5 = 1$: Transfer (X_o) to X

$k_6 = 1$: Transfer (Y_o) to Y

$k_7 = 1$: Transfer (Z_o) to Z

$k_8 = 0$: (always zero)

$k_9 = 1$: Transfer (I_o) to I

$k_{10} = 1$: Transfer (J_o) to J

$k_{11} = 1$: Transfer (K_o) to K

APPENDIX IV

A MICROPROGRAMMING EXAMPLE

This Appendix shows an example of the microprograms required to execute the RNI and EXEC cycles for one main instruction. The timing of the similar program cycles is treated in detail and the notation used to present these programs is explained.

The time required to perform a given operation in a computer is generally dependent on the memory access time, therefore it is common to specify operation time in terms of the memory access or cycle time. In this computer there are two memories and it has been assumed that the Control Memory cycle time is one-tenth that of the Main Memory. To differentiate between the two different cycle times, the terms major cycle and minor cycle are used. One major cycle equals ten minor cycles. Thus to read a cell in M requires a time equal to ten minor cycles or, as more commonly stated, ten minor cycles. Likewise to read a cell in N requires one minor cycle.

To perform a microinstruction, it must be first transferred from N to the decoding register F and at this same time the Control Memory Address Register, H, is incremented by one, preparatory to reading the next instruction. This is the microprogram RNI cycle, which is implemented by hardwired logic. This cycle is understood to occur between each microinstruction and is shown symbolically below.

$$(N\langle H\rangle) \longrightarrow F,\ (H) + 1 \longrightarrow H.$$

This, be definition, requires one minor cycle. The time required to decode and execute a microinstruction is considered to require one minor cycle provided it doesn't involve a transfer to or from Main Memory. If this is required, then the microinstruction is considered

to require one major cycle.

In the following example and all subsequent program listings the Operation Time given is in terms of minor cycles and includes the RNI cycle.

For the purpose of the example, it is assumed that a block of data is to be transferred from a location in M to a location in N. The main instruction opcode for this operation will be TRNFR. The "b" field will contain the starting address of the block in M, the "c" field will give the number of words in the block, and the "d" field will specify the starting address in N.

The first microprogram shown will perform the RNI cycle. This program is the same one intended for use in the computer and will be stored in the first three cells (000-002) of Control Memory. The program is listed in terms of symbolic statements since no formal mnemonics are associated with these microinstructions. All operations listed in the same cell occur simultaneously.

As an aid to understanding the symbolic notation, the following list shows the computer status at the end of this (RNI) microprogram.

Register	Contents
C	Address of next instruction
F	Last microinstruction of the RNI microprogram
H	Address of the entry point for the TRNFR microprogram
R	"a", "i", and "b" fields of the main instruction
X	The starting address in M of the data block
Y	The number of words in the data block
Z	The starting address in N of the data block

The TRNFR microprogram performs the main program EXEC cycle. First the number of words in the data block is put in counter I, then a loop is used to transfer each word via the A register. Each time through the loop, the address registers are incremented, and the counter decremented and sensed for zero. At the end of the transfer, the program jumps to cell #000 to commence the next main program RNI cycle.

All numbers shown in the program listing other than the times are in octal.

RNI MICROPROGRAM

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
11	000		$(M\langle C\rangle) \longrightarrow R$, $(C) + 1 \longrightarrow C$
11	001		$(A_1 \ldots A_8) \longrightarrow Y$, $(A_9 \ldots A_{16}) \longrightarrow Z$, $(C) + 1 \longrightarrow C$, $(M\langle C\rangle) \longrightarrow R$
2	002		$(A_1 \ldots A_5) \longrightarrow H$, $(A_9 \ldots A_{16}) \longrightarrow X$, $(C) + 1 \longrightarrow C$

TRNFR MICROPROGRAM

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
2	15	JPU 0 100	$(100) \longrightarrow H$
2	100	LDA 1 012	$(Y) \longrightarrow A$
2	101	STN 0 I	$(A) \longrightarrow I$
11	102	LDA 1 001	$(M\langle X\rangle) \longrightarrow A$
11	103	STA 0 040	$(A) \longrightarrow N\langle Z\rangle$
2	104	AOD 0 006	$(X) + 1 \longrightarrow X$, $(Z) + 1 \longrightarrow Z$
2	105	SOD 0 100	$(I) - 1 \longrightarrow I$
2	106	NJP 4 302	$I \neq 0$: $102 \longrightarrow H$
2	107	JPU 0 000	$0 \longrightarrow H$

APPENDIX V

MATRIX ARITHMETIC PROGRAMS

This Appendix contains a listing of the programs written for the microprogrammed computer and the 1830 computer which allow both computers to perform the same basic matrix operations. Throughout the program listings, a matrix operation is denoted by the generalized algebraic equations or transformations shown below. A, B, and C denote any matrix with the necessary conformability assumed, and their use as operands is understood to mean the address of the first cell of the memory block containing the matrix elements.

MATAD	$A + B = C$
MATSB	$A - B = C$
MATPS	$A \longrightarrow A^T$
MATML	$A \times B = C$
MATIN	$A \longrightarrow A^{-1}$

The notation used in presenting the microprograms for the microprogrammed computer is the same as in the example program in Appendix IV. It will be noted that there is only one microprogram for MATAD and MATSB. This was possible since the "bookkeeping" was the same for both operations, and only the arithmetic operation was different. The microprogram in general is a straight-forward implementation of the following algorithm.

$$C_{ij} = A_{ij} \pm B_{ij}$$

The MATPS microprogram is also straight-forward and uses the algorithm, $B_{ij} = A_{ji}$ where $B = A^T$.

The MATML operation was accomplished in a slightly different manner than usual. Since each element in a matrix is referenced more

than once in the course of a matrix multiplication, it was decided to transfer both operand matrices into the Control Memory prior to beginning the multiplication, since the access time there is one-tenth that of the Main Memory. Further, it was found advantageous to transpose the B matrix while in the course of transferring it to N. Therefore the MATML microprogram uses a portion of the MATPS microprogram as a microsubroutine.

The matrix inversion microprogram also transfers the operand to N, augmenting it in the process, and then uses the Gauss-Jordan elimination method to accomplish the inversion. Finally, the elements of the inverted matrix are transferred back to M.

To provide the necessary multiply and divide capabilities, two microsubroutines were written using the "shift and add" and "shift and subtract" methods respectively. The starting addresses of these microsubroutines are assumed to be #300 for Multiply and #330 for Divide.

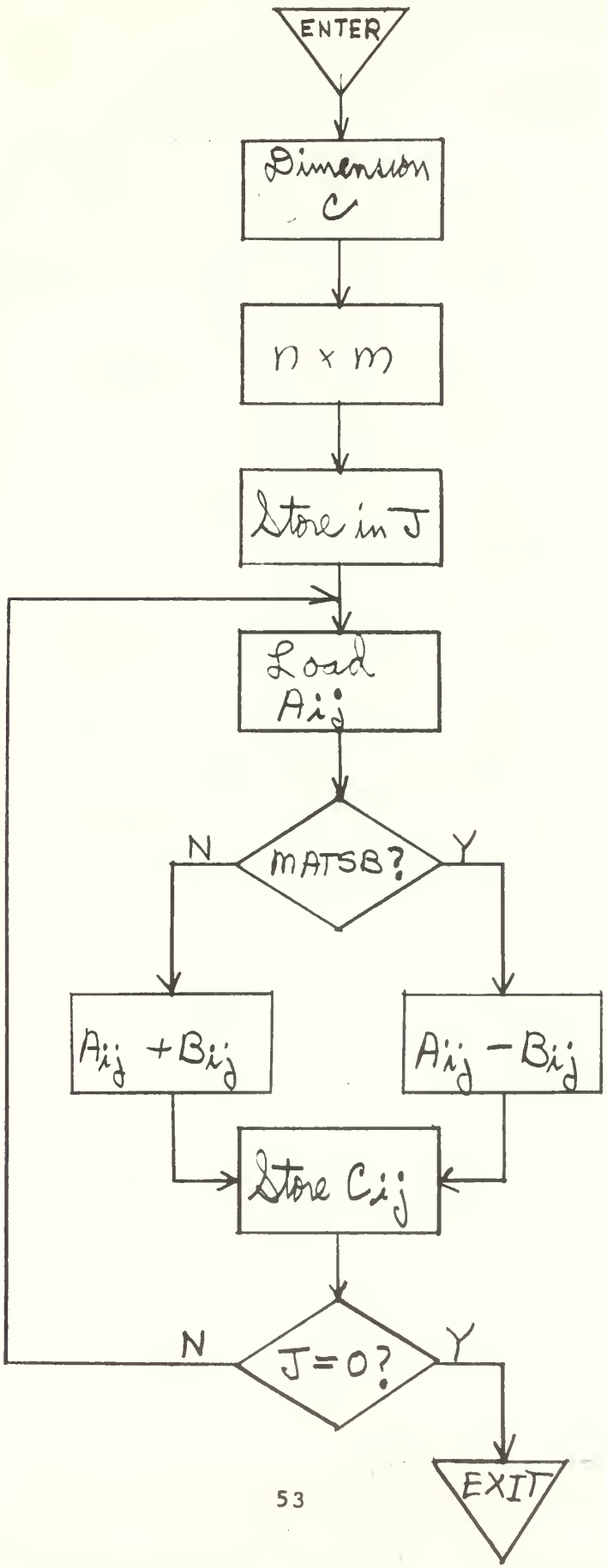
It will be noted that all microprograms are shown as starting in cell #100 of the control Memory. Of course, each program would normally be located in its own section of Control Memory, but it was found at this point that the memory, as originally designed, was not large enough to contain all the microprograms. Rather than redesign the entire computer, the fact is recognized and the microprograms are presented in this general manner.

The programs for use with the 1830 are presented in the format of subroutines. This was done to maintain the similarity with the microprograms, and generally these programs are available in this form for most general-purpose computers. Also included at the beginning of each

subroutine, are the instructions which must appear in the main program to "call" the subroutine. The general method of performing the various matrix operations is the same as used in the microprograms with the exception of matrix multiplication. Here the usual algorithm stated earlier is implemented directly. The use of parentheses in the mnemonic form of instructions indicates those operand values which must be inserted by the subroutine itself based on the entering arguments.

Preceding each microprogram and subroutine is a brief flowchart to assist the interpretation and understanding of each program.

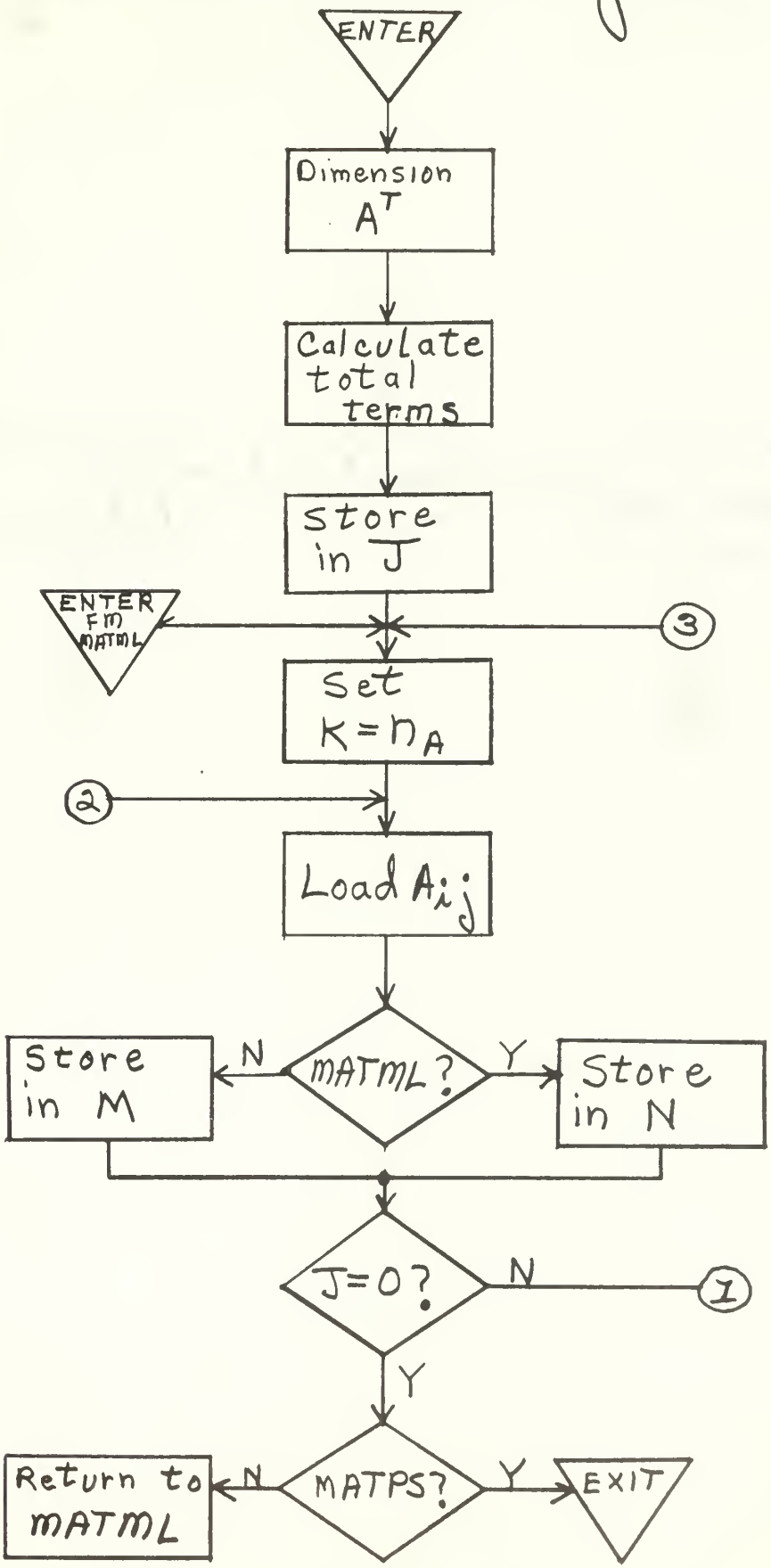
MATAD + MATSB Microprogam

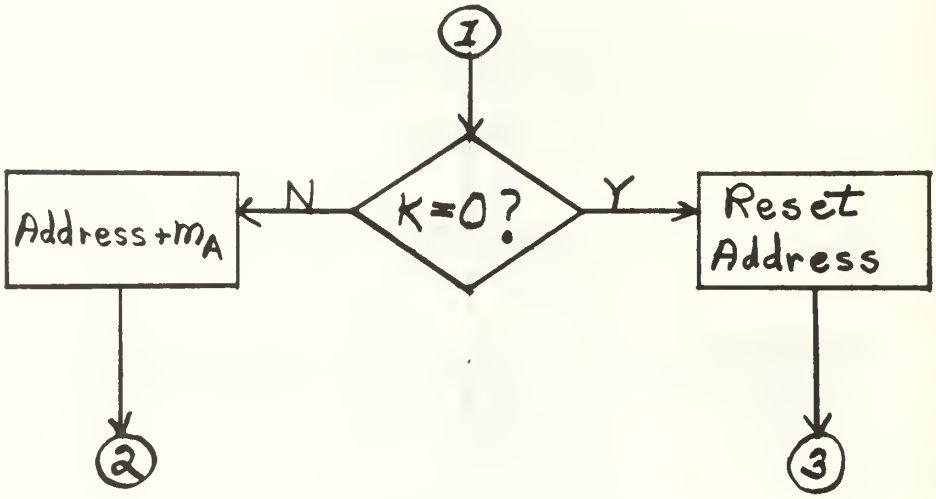


MATAD and MATSB Microprogram

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
11	100	LDA 1 001	(M<X>) ⟶ A
11	101	STA 0 402	(A) ⟶ M<Z>, (A) ⟶ S
2	102	AOD 0 007	(X) + 1 ⟶ X, (Y) + 1 ⟶ Y, (Z)+1 ⟶ Z
11	103	LDA 1 002	(M<Y>) ⟶ A
11	104	STA 0 401	(A) ⟶ Q, (A) ⟶ M<Z>
2	105	LDA 3 060	110 ⟶ A
2	106	SAL 0 303	$(A_L) \longrightarrow 303_L$
56		(Multiply microsubroutine)	
2	107	JPU 0 300	300 ⟶ H
2	110	STN 0 J	(A) ⟶ J
2	111	AOD 0 007	(X)+1 ⟶ X, (Y)+1 ⟶ Y, (Z)+1 ⟶ Z
11	112	LDA 1 001	(M<X>) ⟶ A
2	113	JPI 2 066	(i) = 2: 116 ⟶ H
11	114	ADD 1 002	(A) + (M<Y>) ⟶ A
2	115	JPU 0 067	117 ⟶ H
11	116	SUB 1 002	(A) - (M<Y>) ⟶ A
11	117	STA 0 400	(A) ⟶ M<Z>
2	120	SOD 0 040	(J) - 1 ⟶ J
2	121	NJF 2 111	J ≠ 0: 111 ⟶ H
2	122	JPU 0 000	000 ⟶ H

MATPS Microprogram



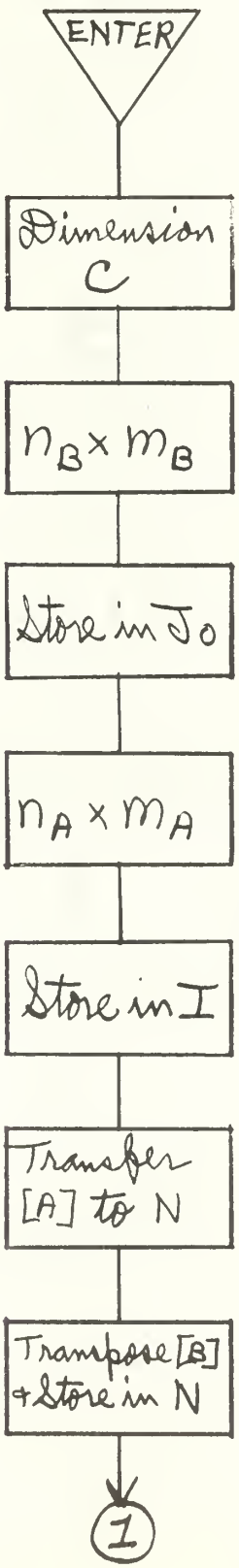

1
K=0?
N
Y
Address + m_A
Reset Address
2
3

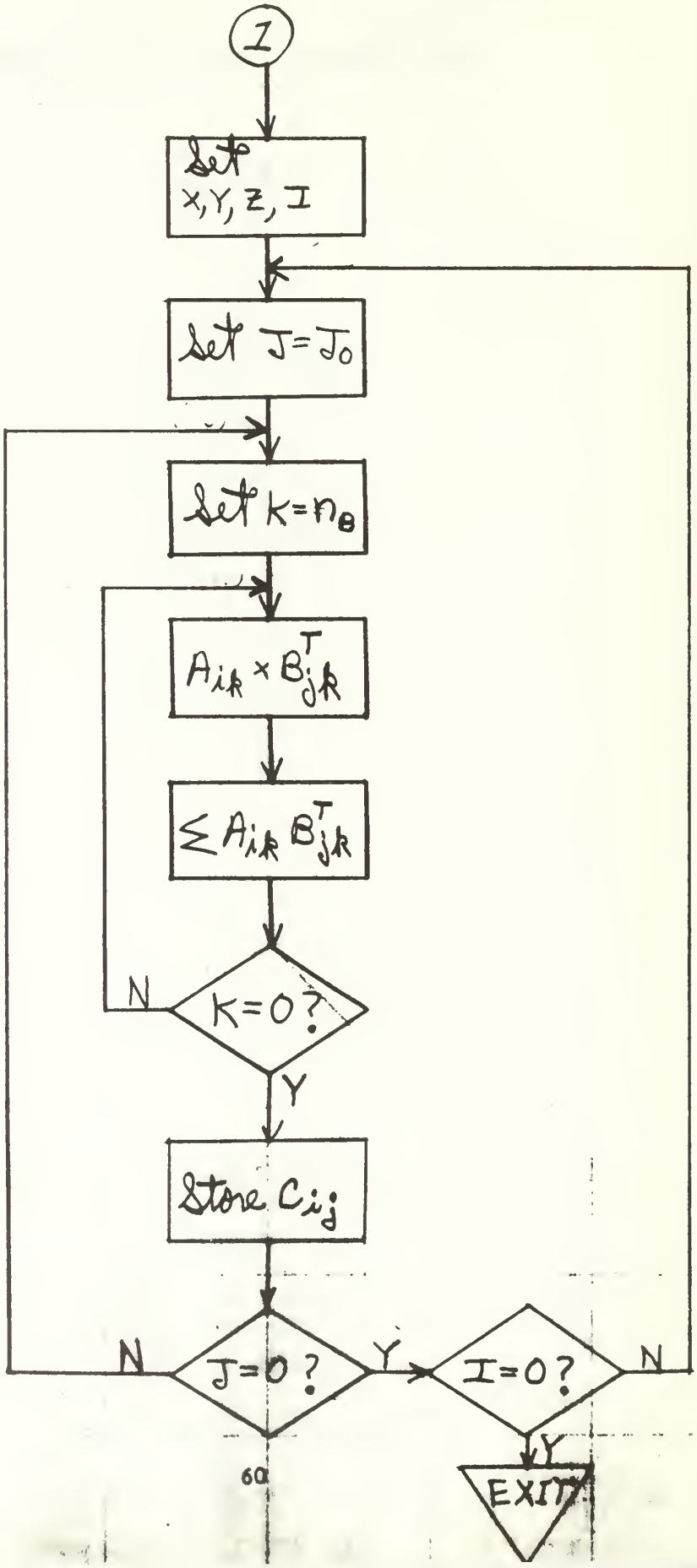
MATPS Microprogram

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
2	100	LDA 1 013	(Z) $\longrightarrow$ A
2	101	STN 0 Z_o	(A) $\longrightarrow$ Z_o
2	102	AOD 0 007	(X) + 1 $\rightarrow$ X, (Y)+1 $\longrightarrow$ Y, (Z)+1 $\longrightarrow$ Z
11	103	LDA 1 001	(M$\langle$X$\rangle$) $\longrightarrow$ A
11	104	STA 0 402	(A) $\longrightarrow$ S, (A) $\longrightarrow$ M$\langle$Z$\rangle$
2	105	STN 0 K_o	(A) $\longrightarrow$ K_o
2	106	AOD 0 004	(X) + 1 $\longrightarrow$ X
2	107	RXF 0 020	(Z_o) $\longrightarrow$ Z
11	110	LDA 1 001	(M$\langle$X$\rangle$) $\longrightarrow$ A
2	111	STN 0 W	(A) $\longrightarrow$ W
11	112	STA 0 401	(A) $\longrightarrow$ Q, (A) $\longrightarrow$ M$\langle$Z$\rangle$
2	113	AOD 0 001	(Z) + 1 $\longrightarrow$ Z
2	114	AOD 0 005	(X) + 1 $\longrightarrow$ X, (Z) + 1 $\longrightarrow$ Z
2	115	LDA 3 095	120 $\longrightarrow$ A
2	116	SAL 0 303	$A_L \longrightarrow 303_L$
2	117	JPU 0 300	300 $\longrightarrow$ H
56		(Multiply microsubroutine)	
2	120	STN 0 J	(A) $\longrightarrow$ J
2	121	LDA 1 011	(X) $\longrightarrow$ A
2	122	STN 0 X_o	(A) $\longrightarrow$ X_o
2	123	RXF 0 001	(K_o) $\longrightarrow$ K
2	124	SOD 0 020	(K) - 1 $\longrightarrow$ K
11	125	LDA 1 001	(M$\langle$X$\rangle$) $\longrightarrow$ A
2	126	JPI 2 106	(i) = 2: 131 $\longrightarrow$ H

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
11	127	STA 0 400	(A)$\longrightarrow$M⟨Z⟩
2	130	JPU 0 107	132$\longrightarrow$H
11	131	STA 0 040	(A)$\longrightarrow$N⟨Z⟩
2	132	SOD 0 040	(J) - 1$\longrightarrow$J
2	133	ZJP 2 125	J = 0: 150$\longrightarrow$H
2	134	ZJP 1 117	K = 0: 142$\longrightarrow$H
2	135	AOD 0 001	(Z) + 1$\longrightarrow$Z
2	136	LDA 1 011	(X)$\longrightarrow$A
2	137	ADD 3 W	(A) + (W)$\longrightarrow$A
2	140	STA 0 020	(A)$\longrightarrow$X
2	141	JPU 0 101	124$\longrightarrow$H
2	142	LDA 2 X_o	(X_o)$\longrightarrow$A
2	143	AOD 0 101	(A) + 1$\longrightarrow$A
2	144	STN 0 X_o	(A)$\longrightarrow$$X_o$
2	145	RXF 0 100	(X_o)$\longrightarrow$X
2	146	AOD 0 001	(Z) + 1$\longrightarrow$Z
2	147	JPU 0 100	123$\longrightarrow$H
2	150	JPI 1 0	(i) = 1: 0$\longrightarrow$H
2	151	JPU 0	($\overset{\text{MATML}}{145}$)$\longrightarrow$H

MATML Microprogram




I
Set
x, Y, z, I
Set J=J0
Set K=nB
Aik × BTjk
Σ Aik BTjk
K=0?
N
Y
Store Cij
J=0?
N
Y
I=0?
N
Y
EXIT

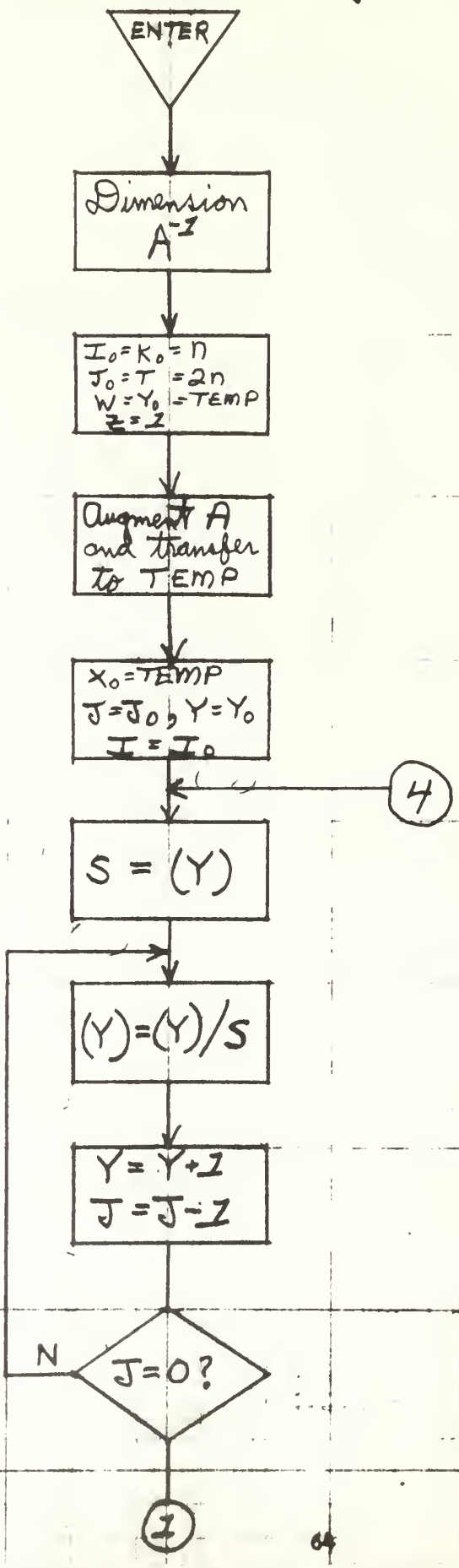
MATML Microprogram

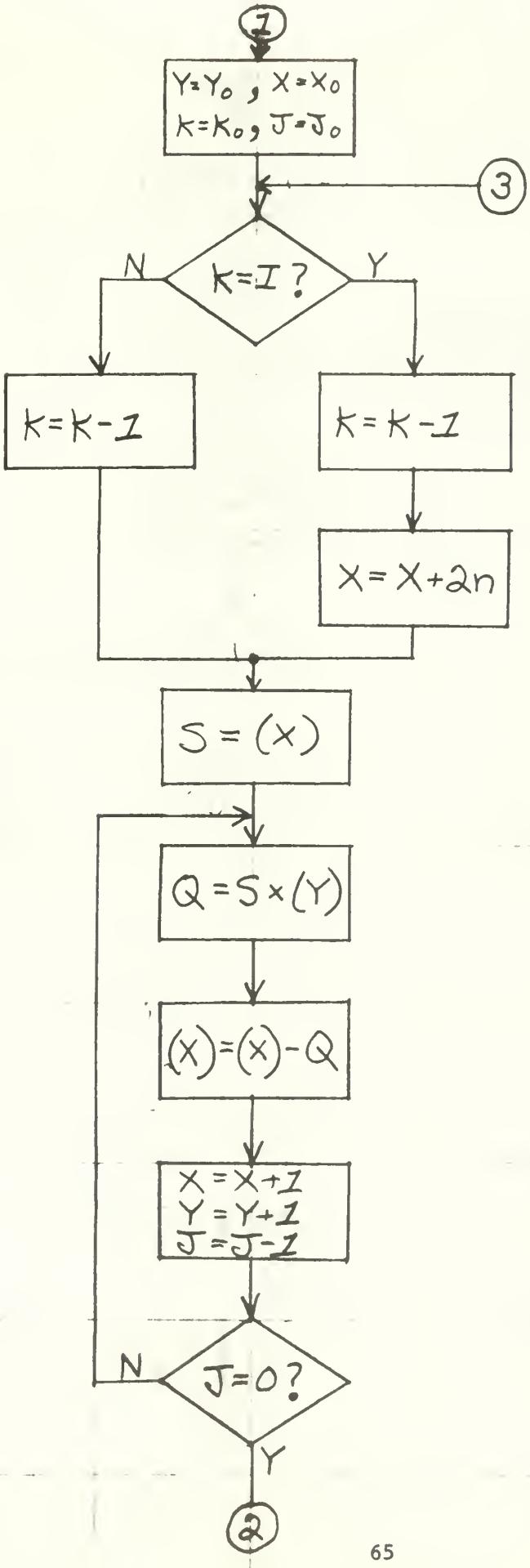
Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
11	101	LDA 1 001	(M⟨X⟩) ⟶ A
11	102	STA 0 400	(A) ⟶ M⟨Z⟩
2	103	STN 0 I_o	(A) ⟶ I_o
11	104	LDA 1 002	(M⟨Y⟩) ⟶ A
2	105	STN 0 K_o	(A) ⟶ K_o
2	106	STA 0 002	(A) ⟶ S
2	107	AOD 0 007	(X) + 1 ⟶ X, (Y)+1 ⟶ Y, (Z)+1 ⟶ Z
11	110	LDA 1 002	(M⟨Y⟩) ⟶ A
11	111	STA 0 401	(A) ⟶ M⟨Z⟩, (A) ⟶ Q
2	112	STN 0 W	(A) ⟶ W
2	113	LDA 3 145	116 ⟶ A
2	114	SAL 0 303	(A_L) ⟶ 303_L
2	115	JPU 0 300	300 ⟶ H
56		(Mulitply microsubroutine)	
2	116	STN 0 J_o	(A) ⟶ J_o
2	117	LDA 2 I_o	(I_o) ⟶ A
2	120	STA 0 001	(A) ⟶ Q
2	121	LDA 3 153	124 ⟶ A
2	122	SAL 0 303	(A_L) ⟶ 303_L
2	123	JPU 0 300	300 ⟶ H
56		(Multiply microsubroutine)	
2	124	STN 0 I	(A) ⟶ I
2	125	LDA 1 13	(Z) ⟶ A
2	126	STN 0 Z_o	(A) ⟶ Z_o
2	127	LDA 3 TEMP	TEMP ⟶ A

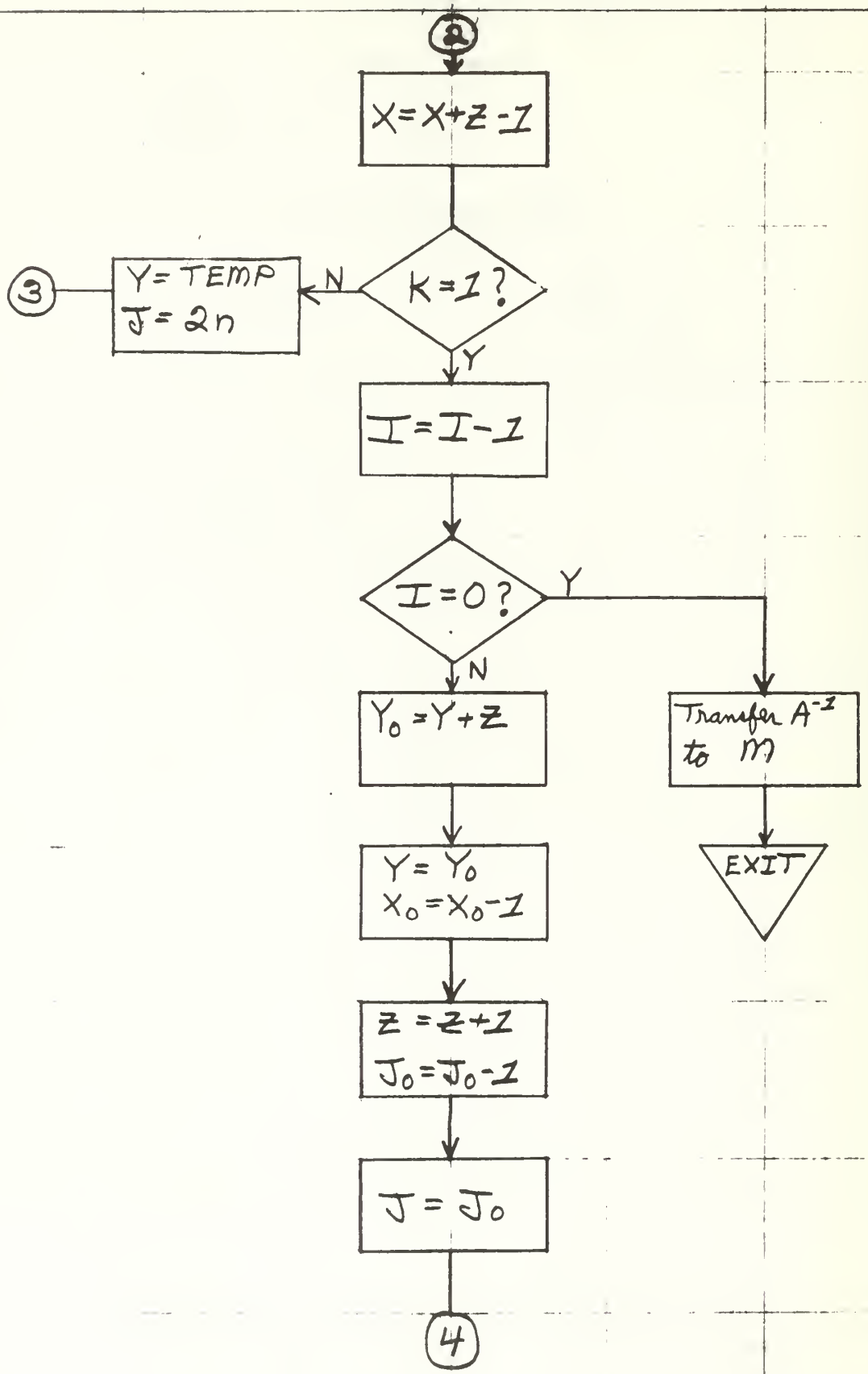
Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
2	130	STA 0 004	$(A) \longrightarrow Z$
2	131	STN 0 T	$(A) \longrightarrow T$
2	132	AOD 0 006	$(X) + 1 \longrightarrow X$, $(Y) + 1 \longrightarrow Y$
11	133	LDA 1 001	$(M\langle X\rangle) \longrightarrow A$
2	134	SOD 0 100	$(I) - 1 \longrightarrow I$
2	136	AOD 0 060	$(X) + 1 \longrightarrow X$, $(Z) + 1 \longrightarrow Z$
2	137	NJP 4 162	$I \neq 0$: $133 \longrightarrow H$
2	140	LDA 1 013	$(Z) \longrightarrow A$
2	141	STN 0 Y_o	$(A) \longrightarrow Y_o$
2	142	LDA 3 174	$145 \longrightarrow A$
2	143	SAL 0 $\binom{\text{MATPS}}{151}$	$(A_L) \longrightarrow$ (MATPS 151)
2	144	JPU 0 100	$\binom{\text{MATPS}}{123} \longrightarrow H$
t		(Transpose microsubroutine)	
2	145	LDA 2 T	$(T) \longrightarrow A$
2	146	STN 0 X_o	$(A) \longrightarrow X_o$
2	147	LDA 3 000	$0 \longrightarrow A$
2	150	STN 0 T	$(A) \longrightarrow T$
2	151	RXF 0 164	$(X_o) \longrightarrow X, (Y_o) \longrightarrow Y, (Z_o) \longrightarrow Z$ $(I_o) \longrightarrow I$
2	152	RXF 0 002	$(J_o) \longrightarrow J$
2	153	RXF 0 001	$(K_o) \longrightarrow K$
2	154	AOD 0 001	$(Z) + 1 \longrightarrow Z$
2	155	AOD 0 006	$(X) + 1 \longrightarrow X$, $(Y) + 1 \longrightarrow Y$
2	156	LDA 1 005	$(N\langle X\rangle) \longrightarrow A$
2	157	STA 0 002	$(A) \longrightarrow S$
2	160	LDA 1 006	$(N\langle Y\rangle) \longrightarrow A$

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
2	161	STA 0 001	(A)⟶Q
2	162	LDA 3 194	165⟶A
2	163	SAL 0 303	(A_L)⟶303L
2	164	JPU 0 300	300⟶H
56		(Multiply subroutine)	
2	165	ADD 2 T	(A) + (T)⟶A
2	166	STN 0 T	(A)⟶T
2	167	SOD 0 060	(K) - 1⟶K, (J) - 1⟶J
2	170	NJP 1 184	K ≠ 0: 155⟶H
2	171	LDA 2 T	(T)⟶A
11	172	STA 0 400	(A)⟶M⟨Z⟩
2	173	ZJP 2 207	J = 0: 200⟶H
2	174	LDA 1 011	(X)⟶A
2	175	SUB 2 K_o	(A) - (K_o)⟶A
2	176	STA 0 020	(A)⟶X
2	177	JPU 0 182	153⟶H
2	200	SOD 0 100	(I) - 1⟶I
2	201	ZJP 4 000	I = 0: 0⟶H
2	202	LDA 1 012	(Y)⟶A
2	203	SUB 2 J_o	(A) - (J_o)⟶A
2	204	STA 0 010	(A)⟶Y
2	205	JPU 0 181	152⟶H

MATIN Microprogram




1
Y=Y0 , X=X0
K=K0, J=J0
3
K=I ?
N
Y
K=K-I
K=K-I
X=X+2n
S=(X)
Q=S×(Y)
(X)=(X)-Q
X=X+I
Y=Y+I
J=J-I
J=0?
N
Y
2


2
X=X+Z-I
K=I?
N
3
Y=TEMP
J=2n
Y
I=I-1
I=0?
Y
N
Transfer A^-1 to M
EXIT
Y0=Y+Z
Y=Y0
X0=X0-I
Z=Z+I
J0=J0-I
J=J0
4

MATIN MICROPROGRAM

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
11	100	LDA 1 001	$(M\langle X\rangle) \longrightarrow A$
2	101	STN 0 I_o	$(A) \longrightarrow I_o$
2	102	STN 0 K_o	$(A) \longrightarrow K_o$
11	103	STA 0 400	$(A) \longrightarrow M\langle Z\rangle$
2	104	ADD 2 I_o	$(A) + (I_o) \longrightarrow A$
2	105	STN 0 J_o	$(A) \longrightarrow J_o$
2	106	STN 0 T	$(A) \longrightarrow T$
2	107	AOD 0 003	$(X) + 1 \longrightarrow X$, $(Z) + 1 \longrightarrow Z$
11	110	LDA 1 001	$(M\langle X\rangle) \longrightarrow A$
11	111	STA 0 400	$(A) \longrightarrow M\langle Z\rangle$
2	112	LDA 1 013	$(Z) \longrightarrow A$
2	113	STN 0 Z_o	$(A) \longrightarrow Z_o$
2	114	LDA 3 001	$1 \longrightarrow A$
2	115	STA 0 004	$(A) \longrightarrow Z$
2	116	LDA 3 TEMP	$TEMP \longrightarrow A$
2	117	STN 0 W	$(A) \longrightarrow W$
2	120	STN 0 Y_o	$(A) \longrightarrow Y_o$
2	121	SOD 0 010	$(A) - 1 \longrightarrow A$
2	122	STA 0 010	$(A) \longrightarrow Y$
2	123	RXF 0 001	$(K_o) \longrightarrow K$
2	124	RXF 0 006	$(I_o) \longrightarrow I$, $(J_o) \longrightarrow J$
2	125	AOD 0 006	$(X) + 1 \longrightarrow X$, $(Y) + 1 \longrightarrow Y$
11	126	LDA 1 001	$(M\langle X\rangle) \longrightarrow A$
2	127	STA 0 100	$(A) \longrightarrow N\langle Y\rangle$

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
2	130	SOD 0 140	(I) - 1⟶I, (J) - 1⟶J
2	131	NJP 4 134	I ≠ 0 : 125⟶H
2	132	LDA 2 K	(K)⟶A
2	133	SUB 2 J	(A) - (J)⟶A
2	134	AJP 1 137	A = 0 : 137⟶H
2	135	LDA 3 000	0⟶A
2	136	JPU 0 140	140⟶H
2	137	LDA 3 001	1⟶A
2	140	AOD 0 002	(Y) + 1⟶Y
2	141	STA 0 100	A⟶N⟨Y⟩
2	142	SOD 0 040	(J) - 1⟶J
2	143	NJP 2 132	J ≠ 0 : 132⟶H
2	144	SOD 0 020	(K) - 1⟶K
2	145	NJP 1 124	K ≠ 0 : 124⟶H
2	146	LDA 2 Y_o	(Y_o)⟶A
2	147	STN 0 X_o	(A)⟶X_o
2	150	RXF 0 046	(Y_o)⟶Y, (J_o)⟶J, (I_o)⟶I
2	151	LDA 1 006	(N⟨Y⟩)⟶A
2	152	STA 0 002	(A)⟶S
2	153	LDA 1 006	(N⟨Y⟩)⟶A
2	154	LDA 3 143	143⟶A
2	155	SAL 0 333	(A_L)⟶333_L
2	156	JPU 0 330	330⟶H
68		(Divide microsubroutine)	
2	157	LDA 1 016	(Q)⟶A

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
2	160	STA 0 100	(A)⟶N⟨Y⟩
2	161	SOD 0 040	(J) - 1⟶J
2	162	AOD 0 002	(Y) + 1⟶Y
2	163	NJP 2 153	J ≠ 0 : 153⟶H
2	164	RXF 0 143	(Y_o)⟶Y, (J_o)⟶J, (K_o)⟶K, (X_o)⟶X
2	165	LDA 2 K	(K)⟶A
2	166	SUB 2 I	(A) - (I)⟶A
2	167	SOD 0 020	(K) - 1⟶K
2	170	AJP 2 174	A ≠ 0 : 174⟶H
2	171	LDA 1 011	(X)⟶A
2	172	ADD 2 T	(A) + (T)⟶A
2	173	STA 0 020	(A)⟶X
2	174	LDA 1 005	(N⟨X⟩)⟶A
2	175	STA 0 002	(A)⟶S
2	176	LDA 1 006	(N⟨Y⟩)⟶A
2	177	STA 0 001	(A)⟶Q
2	200	LDA 3 203	203⟶A
2	201	SAL 0 303	(A_L)⟶303_L
2	202	JPU 0 300	300⟶H
56		(Multiply subroutine)	
2	203	STA 0 001	(A)⟶Q
2	204	LDA 1 005	N ((X))⟶A
2	205	SUB 1 016	(A) - (Q)⟶A
2	206	STA 0 200	(A)⟶N⟨X⟩
2	207	SOD 0 040	(J) - 1⟶J

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
2	210	AOD 0 006	(X) + 1⟶X, (Y) + 1⟶Y
2	211	NJP 2 176	J ≠ 0 : 176⟶H
2	212	LDA 1 013	(Z)⟶A
2	213	SOD 0 010	(A) - 1⟶A
2	214	ADD 1 011	(A) + (X)⟶A
2	215	STA 0 020	(A)⟶X
2	216	LDA 3 001	1⟶A
2	217	SUB 2 K	(A) - (K)⟶A
2	220	AJP 1 223	A = 0 : 223⟶H
2	221	RXF 0 012	(Y_o)⟶Y, (J_o)⟶J
2	222	JPU 0 165	165⟶H
2	223	SOD 0 100	(I) - 1⟶I
2	224	ZJP 4 241	I = 0 : 241⟶H
2	225	LDA 1 012	(Y)⟶A
2	226	ADD 1 013	(A) + (Z)⟶A
2	227	STN 0 Y_o	(A)⟶Y_o
2	230	RXF 0 040	(Y_o)⟶Y
2	231	LDA 2 X_o	(X_o)⟶A
2	232	AOD 0 011	(A) + 1⟶A, (Z) + 1⟶Z
2	233	STN 0 X_o	(A)⟶X_o
2	234	LDA 2 J_o	(J_o)⟶A
2	235	SOD 0 010	(A) - 1⟶A
2	236	STN 0 J_o	(A)⟶J_o
2	237	RXF 0 002	(J_o)⟶J
2	240	JPU 0 151	151⟶H

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
2	241	LDA 2 I_o	$(I_o) \longrightarrow A$
2	242	STN 0 J_o	$(A) \longrightarrow J_o$
2	243	RXF 0 023	$(Z_o) \longrightarrow Z$, $(K_o) \longrightarrow K$, $(J_o) \longrightarrow J$
2	244	AOD 0 001	$(Z) + 1 \longrightarrow Z$
2	245	LDA 2 W	$(W) \longrightarrow A$
2	246	ADD 2 I_o	$(A) + (I_o) \longrightarrow A$
2	247	STA 0 020	$(A) \longrightarrow X$
2	250	LDA 1 005	$(N\langle X \rangle) \longrightarrow A$
2	251	STA 0 020	$(A) \longrightarrow M\langle Z \rangle$
2	252	AOD 0 005	$(X) + 1 \longrightarrow X$, $(Z) + 1 \longrightarrow Z$
2	253	SOD 0 040	$(J) - 1 \longrightarrow J$
2	254	NJP 2 250	$J \neq 0 : 250 \longrightarrow H$
2	255	SOD 0 020	$(K) - 1 \longrightarrow K$
2	256	ZJP 1 000	$K = 0 : 0 \longrightarrow H$
2	257	RXF 0 002	$(J_o) \longrightarrow J$
2	260	LDA 1 011	$(X) \longrightarrow A$
2	261	JPU 0 246	$246 \longrightarrow H$

MULTIPLY Microsubroutine

Oper Time	Cell #	Contents (Symbolic)
2	300	$0 \longrightarrow Q,\ 16 \longrightarrow G$
2	301	$G \neq 0$: $Q_{16} = 0$: $301 \longrightarrow H,\ (G) - 1 \longrightarrow G,\ (A_1 \ldots Q_{15}) \longrightarrow (A_2 \ldots Q_{10})$ $Q_{16} = 1$: $302 \longrightarrow H,\ (G) - 1 \longrightarrow G$ $G = 0$: $303 \longrightarrow H$
2	302	$(A_1 \ldots A_{16}) + (S) \longrightarrow (A_2 \ldots Q_1),\ (Q_1 \ldots Q_{15}) \longrightarrow (Q_2 \ldots Q_{16}),\ 301 \longrightarrow H$
2	303	$(\ ) \longrightarrow H$

Average Operation Time: 44 minor cycles

Maximum Operation Time: 68 minor cycles

This Multiply microsubroutine is composed of special-purpose instructions which, like the RNI microprogram, have no mnemonic names assigned. This method of implementing a multiplication is necessary to make the multiply time competitive with conventional machines. The G refers to a special 3-bit counter used with the multiply and divide routines exclusively.

The first instruction performs the necessary initialization. The second instruction accomplishes all the steps which must be performed each time, i.e., sensing the right most bit in the Q register for 0 or 1, and decrementing the counter. If $Q_{16} = 0$, the contents of AQ are shifted right one bit and the same instruction is read and executed again. The reading and executing take two minor cycles.

If $Q_{16} = 1$, the next instruction is executed also requiring 2 minor cycles. This instruction adds the multiplicand in S to the A register, shifting the result one bit to the right, and at the same time shifts Q one bit right.

Then a jump back to #301 occurs and the cycle is repeated until the multiplication is complete. At that time instruction in cell #301 senses G = 0, and causes a jump to #303. Cell #303 contains a standard JPU instruction whose operand is set by the calling microprogram and provides the return to that microprogram. Note that the total execution time depends on the number being multiplied. An average and a maximum time are shown. A value of 56 minor cycles was used in calculating program running times.

DIVIDE Microsubroutine

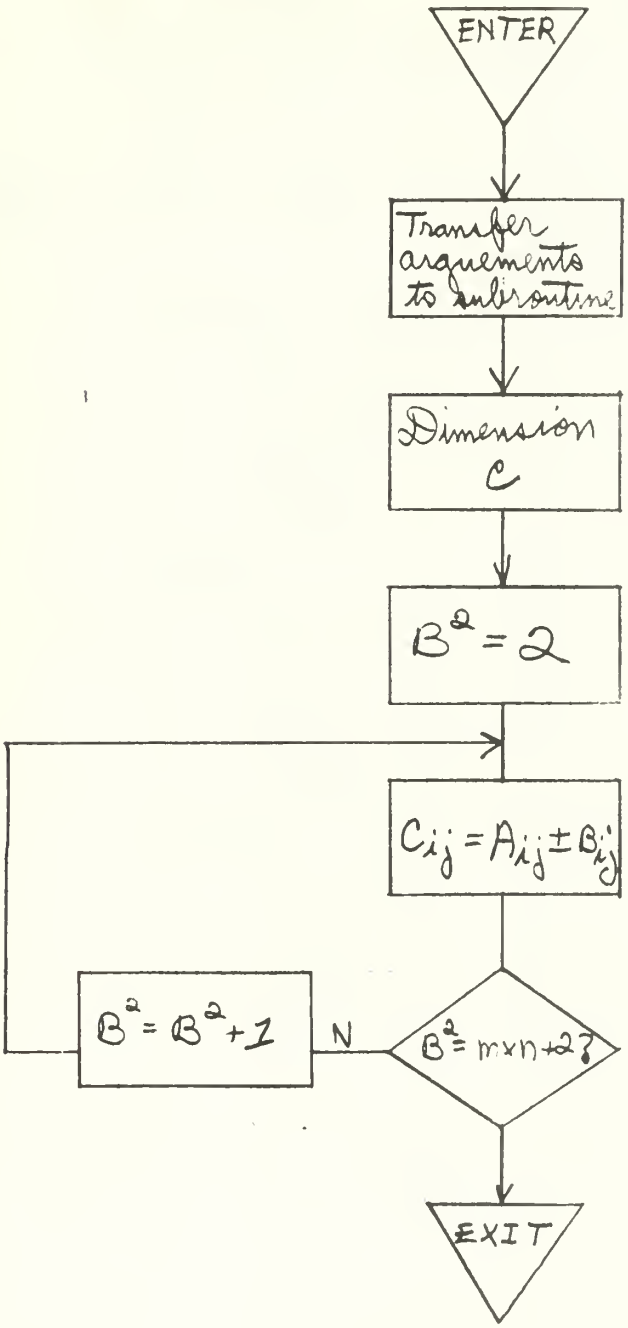
Oper Time	Cell #	Contents (Symbolic)
2	330	$0 \longrightarrow Q,\ 16 \longrightarrow G$
2	331	$(A_2 \ldots Q_1) - (S) \longrightarrow (A_1 \ldots A_{16}),\ (G) - 1 \longrightarrow G$
2	332	$\begin{cases} G \neq 0 : \begin{cases} A > 0 : 1 \longrightarrow Q_{16},\ 331 \longrightarrow H \\ A \not> 0 : (A)+(S) \longrightarrow S,\ 0 \longrightarrow Q_{16},\ 331 \longrightarrow H \end{cases} \\ G = 0 : \quad 333 \longrightarrow H \end{cases}$
2	333	$(\quad) \longrightarrow H$

Operation Time: 68 minor cycles

This Divide microsubroutine is also composed of special-purpose instructions to permit competitive timing.

Again the first instruction performs the initialization. The second instruction performs the equivalent of left shifting AQ one bit and then subtracting S from A all in one operation. The third instruction senses the contents of A and takes the appropriate action to implement the "shift and subtract" procedure. This two instruction cycle is repeated until the division is complete. The G counter is sensed for 0, and when found, a jump to the final instruction occurs. This last instruction is a standard JPU which causes a return to the calling microprogram.

MATAD & MATSB Subroutine

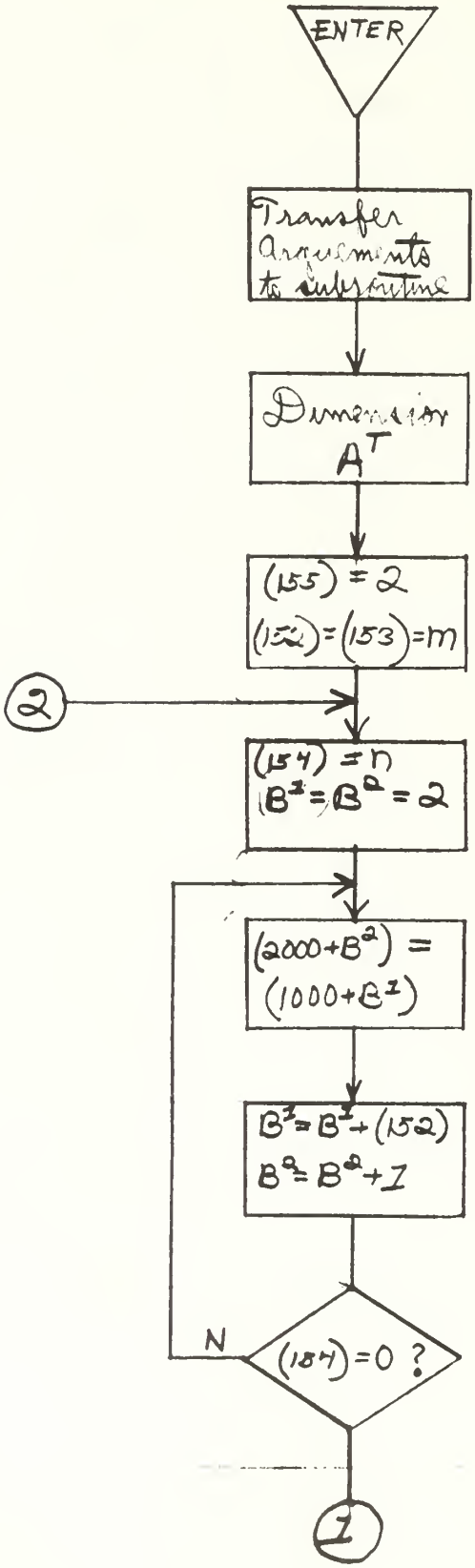


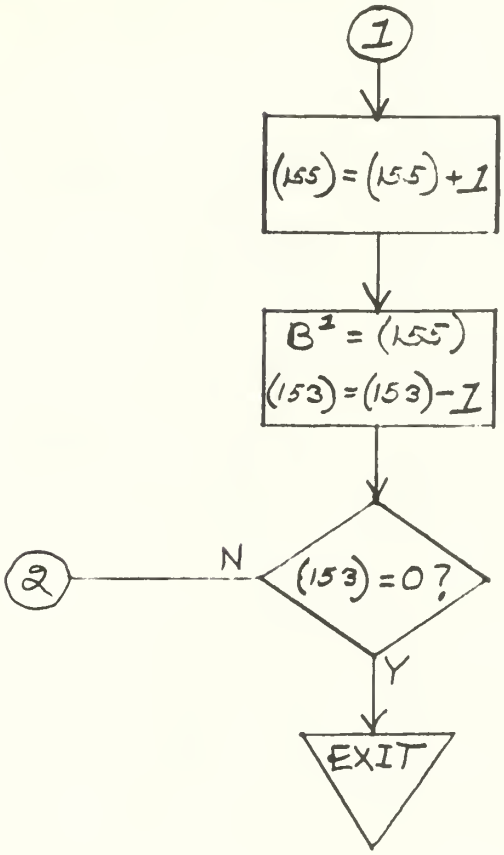
MATAD and MATSB Subroutines

Oper Time	Cell #	Contents (Mnemonic)			Contents (Symbolic)
3	600	RTJ	0 0 0	100	"600" $\longrightarrow 100_L$, 101 $\longrightarrow$ P
-	601	1	0 0 0	2000	A = 1000, B = 2000
-	602	-	- - $^{+}_{-}$1	3000	+1 for Add, -1 for Sub, C=3000
-	100	0	0 0 2	(600)	
3	101	RPL.Y +1	0 1 0	100	$(100_L) + 1 \longrightarrow 100_L$
2	102	ENT.A	0 1 0	100	$(100_L) \longrightarrow$ A
2	103	STR.A	0 1 0	114	(A) $\longrightarrow 114_L$
2	104	STR.A	0 1 0	116	(A) $\longrightarrow 116_L$
3	105	RPL.Y+ 1	0 1 0	100	$(100_L) + 1 \longrightarrow 100_L$
2	106	ENT.A	0 1 0	100	$(100_L) \longrightarrow$ A
2	107	STR.A	0 1 0	121	(A) $\longrightarrow 121_L$
2	110	STR.A	0 1 0	123	(A) $\longrightarrow 123_L$
3	111	RPL.Y +1	0 1 0	100	$(100_L) + 1 \longrightarrow 100_L$
2	112	ENT.A	0 1 0	100	$(100_L) \longrightarrow$ A
2	113	STR.A	0 1 0	133	(A) $\longrightarrow 133_L$
2	114	ENT.A	0 2 0	(601)	$(601_U) \longrightarrow$ A
2	115	STR.A	0 1 0	140	(A) $\longrightarrow 140_L$
2	116	ENT.A	0 1 0	(601)	$(601_L) \longrightarrow$ A
2	117	STR.A	0 1 0	141	(A) $\longrightarrow 141_L$
2	120	STR.A	0 1 0	142	(A) $\longrightarrow 142_L$
2	121	ENT.A	0 2 0	(602)	$(602_U) \longrightarrow$ A
2	122	STR.A	0 1 0	137	(A) $\longrightarrow 137_L$

Oper Time	Cell #	Contents (Mnemonic)			Contents (Symbolic)
2	123	ENT.A	0 1 0	(602)	$(602_L) \longrightarrow A$
2	124	STR.A	0 1 0	143	$(A) \longrightarrow 143$
3	125	ENT.B^n	2 0 0	0000	$0000 \longrightarrow B^2$
2	126	ENT.Q	0 3 2	(1000)	$(1000) \longrightarrow Q$
2	127	STR.Q	0 3 2	(3000)	$(Q) \longrightarrow 3000$
3	130	BSK.B^n	2 0 0	0	$(B^2) + 1 \longrightarrow B^2$
2	131	ENT.A	0 3 2	(1000)	$(1001) \longrightarrow A$
2	132	STR.A	0 3 2	(3000)	$(A) \longrightarrow 3001$
10	133	MUL	0 3 2	(1000)	$(A) \times (1001) \longrightarrow AQ$
2	134	ADD.Q	0 0 0	2	$(Q) + 2 \longrightarrow Q$
2	135	STR.Q	0 1 0	144	$(Q) \longrightarrow 144_L$
3	136	BSK.B^n	2 0 0	0000	$(B^2) + 1 \longrightarrow B^2$
2	137	ENT.Q	0 0 0	(+1 or -1)	$+1$ or $-1 \longrightarrow Q$
2	140	ENT.A	0 0 0	(1000)	$(1002) \longrightarrow A$, Skip NI if Q Neg
2	141	ADD.A	2 3 2	(2000)	$(A)+(2002) \longrightarrow A$, Skip NI if Q Pos
2	142	SUB.A	0 3 2	(2000)	$(A) - (2002) \longrightarrow A$
2	143	STR.A	0 3 2	(3000)	$(A) \longrightarrow 3000$
3	144	BSK.B^n	2 0 0	(m x n+2)	Skip NI if $B^2 = m \times n + 2$, or $(B^2) + 1 \longrightarrow B^2$
2	145	JMP	1 0 0	140	Go to 140
2	146	JMP	1 0 0	(603)	Return to Main Program

MATPS Subroutine



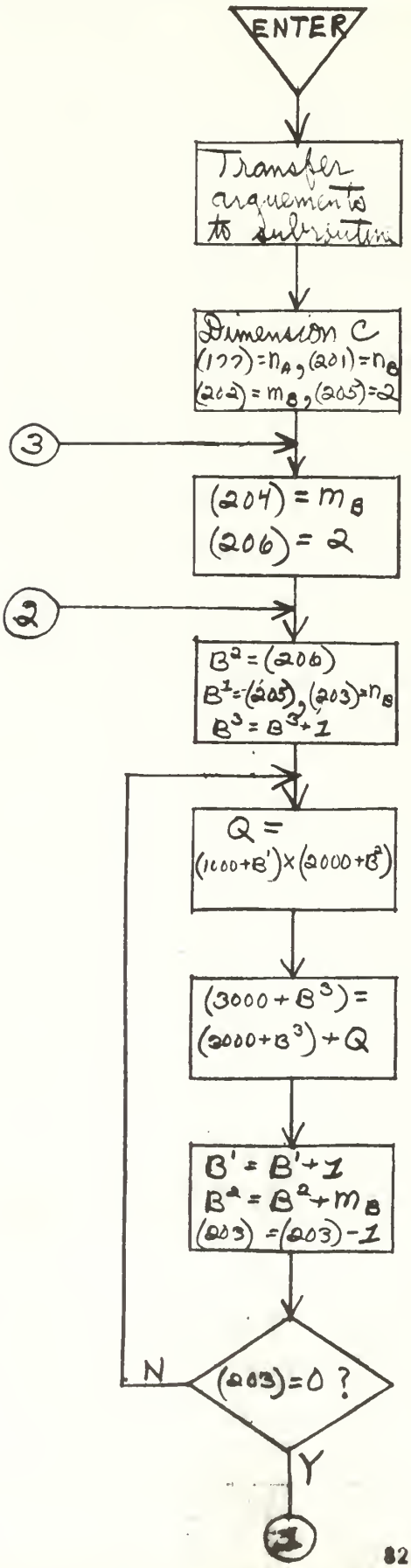

1
(155) = (155) + 1
B¹ = (155)
(153) = (153) - 1
2
N
(153) = 0 ?
Y
EXIT

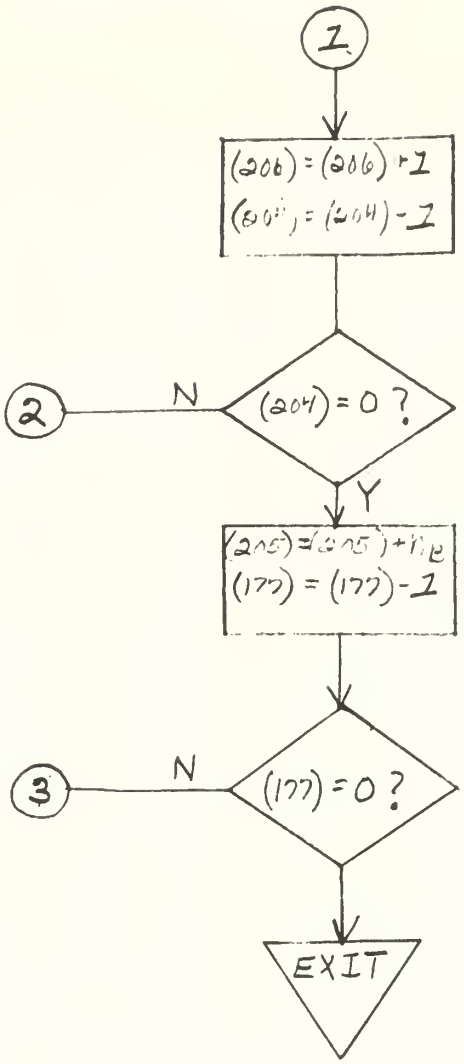
MATPS Subroutine

Oper Time	Cell #	Contents (Mnemonic)				Contents (Symbolic)	
3	600	RTJ	0 0 0	100		"600" $\longrightarrow 100_L$ & "101" $\longrightarrow$ P	
-	601	1	0 0 0	2000		A = 1000, A^T = 2000	
-	100	0	0 0 0	(600)			
3	101	RPL.Y+1	0 1 0	100		$(100_L) + 1 \longrightarrow 100_L$	
2	102	ENT.A	0 1 0	100		$(100_L) \longrightarrow$ A	
2	103	STR.A	0 1 0	110		(A) $\longrightarrow$ 110	
2	104	STR.A	0 1 0	113		(A) $\longrightarrow 113_L$	
3	105	RPL.Y+1	0 1 0	100		$(100_L) + 1 \longrightarrow 100_L$	
2	106	ENT.A	0 1 0	100		$(100_L) \longrightarrow$ A	
2	107	STR.A	0 1 0	151		(A) $\longrightarrow 151_L$	
2	110	ENT.A	0 2 0	(601)		$(601_U) \longrightarrow$ A	
2	111	STR.A	0 1 0	121		(A) $\longrightarrow 121_L$	
2	112	STR.A	0 1 0	135		(A) $\longrightarrow 135_L$	
2	113	ENT.A	0 1 0	(601)		$(601_L) \longrightarrow$ A	
2	114	STR.A	0 1 0	122		(A) $\longrightarrow 122_L$	
2	115	STR.A	0 1 0	136		(A) $\longrightarrow 136_L$	
2	116	ENT.Q	0 1 0	-1		-1 $\longrightarrow$ Q	
3	117	ENT.B^n	1 0 0	0000		0 $\longrightarrow B^1$	
3	120	ENT.B^n	2 0 0	0001		1 $\longrightarrow B^2$	
						1st Pass	2nd Pass
2	121	ENT.A	0 3 1	1000		(1000) $\longrightarrow$ A	(1001) $\longrightarrow$ A
2	122	STR.A	3 3 2	2000		(A) $\longrightarrow$ 2001	(A) $\longrightarrow$ 2000
2	123	STR.A	2 3 0	152		"skip"	(A) $\longrightarrow$ 152
2	124	STR.A	0 0 0	0000		(A) $\longrightarrow$ Q	"skip"
3	125	BSK.B^n	1 0 0	0005		$(B^1)+1 \longrightarrow B^1$	$(B^1)+1 \longrightarrow B^1$

Oper Time	Cell #	Contents (Mnemonic)				Contents (Symbolic)	
						1st Pass	2nd Pass
3	126	BJP.B^n	2	0 0	121	$(B^2)-1 \longrightarrow B^2$ jump to 121	go to NI
3	127	ENT.B^n	2	0 0	2	$2 \longrightarrow B^2$	
2	130	ENT.A	0	0 0	0002	$2 \longrightarrow A$	
2	131	STR.A	0	3 0	155	$(A) \longrightarrow 155$	
2	132	ENT.A	0	3 0	152	$(152) \longrightarrow A$	
2	133	STR.A	0	3 0	153	$(A) \longrightarrow 153$	
2	134	STR.Q	0	3 0	154	$(Q) \longrightarrow 154$	
2	135	ENT.A	0	3 1	(1000)	$(1000 + B^1) \longrightarrow A$	
2	136	STR.A	0	3 2	(2000)	$(A) \longrightarrow 2000 + B^2$	
3	137	STR.B^n	1	7 0	0000	$(B^1) \longrightarrow A$	
2	140	ADD.A	0	3 0	152	$(A) + (152) \longrightarrow A$	
3	141	ENT.B^n	1	7 0	0000	$(A) \longrightarrow B^1$	
3	142	BSK.B^n	2	0 0	0000	$(B^2) + 1 \longrightarrow B^2$	
3	143	RPL.Y-1	0	3 0	154	$(154) - 1 \longrightarrow 154$	
2	144	JP	5	0 0	135	Go to 135 if (A) non zero	
3	145	RPL.Y+1	0	3 0	155	$(155) + 1 \longrightarrow 155$	
3	146	ENT.B^n	1	3 0	155	$(155) \longrightarrow B^1$	
3	147	RPL.Y-1	0	3 0	153	$(153) - 1 \longrightarrow 153$	
2	150	JP	5	0 0	134	Go to 134 if (A) non-zero	
2	151	JP	1	0 0	(602)	Go to 602	
	152 153 154 155	Temporary Storage and Counters					

MATML Subroutine




1
(206) = (206) + I
(204) = (204) - I
2
N
(204) = 0 ?
Y
(205) = (205) + I
(177) = (177) - I
3
N
(177) = 0 ?
EXIT

MATMUL Subroutines

Oper Time	Cell #	Contents (Mnemonic)				Contents (Symbolic)
2	600	RTJ	0	0	100	"6000"$\longrightarrow$100, "101"$\longrightarrow$P
-	601	1	0	0	2000	A = 1000, B = 2000
-	602	1	0	0	3000	C = 3000
-	100	0	0	0	(600)	
3	101	RPL.Y+1	0	1	100	$(100_L) + 1 \longrightarrow 100_L$
2	102	STR.A	0	1	106	(A)$\longrightarrow$106
2	103	STR.A	0	1	111	(A)$\longrightarrow$111
3	104	RPL.Y+1	0	1	100	$(100_L) + 1 \longrightarrow 100_L$
2	105	STR.A	0	1	115	(A)$\longrightarrow$115
2	106	ENT.A	0	2	(601)	$(601_U) \longrightarrow$A
2	107	STR.A	0	1	124	(A)$\longrightarrow$124
2	110	STR.A	0	1	155	(A)$\longrightarrow$155
2	111	ENT.A	0	1	(601)	$(601_L) \longrightarrow$A
2	112	STR.A	0	1	130	(A)$\longrightarrow$130
2	113	STR.A	0	1	133	(A)$\longrightarrow$133
2	114	STR.A	0	1	156	(A)$\longrightarrow$156
2	115	ENT.A	0	1	602	$(602_L) \longrightarrow$A
2	116	STR.A	0	1	125	(A)$\longrightarrow$125
2	117	STR.A	0	1	134	(A)$\longrightarrow$134
2	120	STR.A	0	1	151	(A)$\longrightarrow$151
2	121	STR.A	0	1	160	(A)$\longrightarrow$160

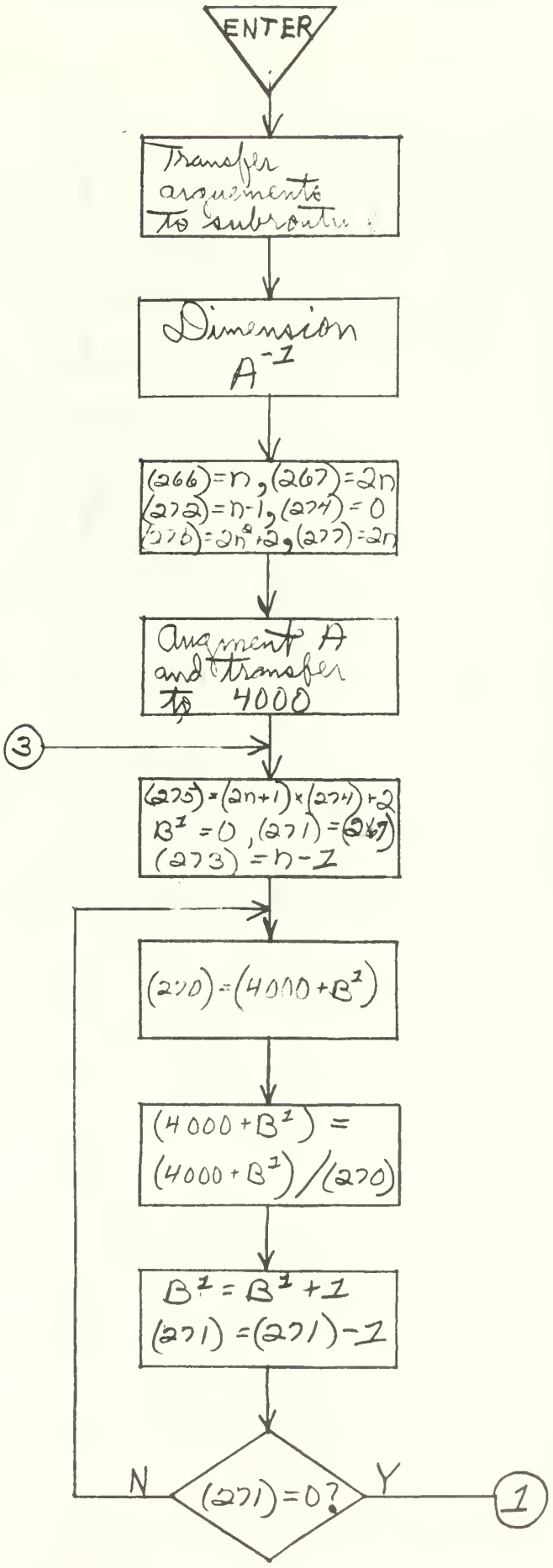
Oper Time	Cell #	Contents (Mnemonic)		Contents (Symbolic)
3	122	ENT.B^n	2 0 0 0000	$0 \longrightarrow B^2$
3	123	ENT.B^n	3 0 0 0000	$0 \longrightarrow B^3$
2	124	ENT.A	0 3 0 (1000)	$(1000) \longrightarrow A$
2	125	STR.A	0 3 0 (3000)	$(A) \longrightarrow 3000$
2	126	STR.A	0 3 0 177	$(A) \longrightarrow 177$
3	127	BSK.B^n	3 0 0 0005	$(B^3) + 1 \longrightarrow B^3$
2	130	ENT.A	0 3 2 (2000)	$(2000) \longrightarrow A$
2	131	STR.A	0 3 0 201	$(A) \longrightarrow 201$
3	132	BSK.B^n	2 0 0 0005	$(B^2) + 1 \longrightarrow B^2$
2	133	ENT.A	0 3 2 (2000)	$(2001) \longrightarrow A$
2	134	STR.A	0 3 3 (3000)	$(A) \longrightarrow 3001$
2	135	STR.A	0 3 0 202	$(A) \longrightarrow 202$
2	136	ENT.A	0 0 0 0002	$2 \longrightarrow A$
2	137	STR.A	0 3 0 205	$(A) \longrightarrow 205$
3	140	RPL.Y + 1	0 1 0 5000	$(100_L) + 1 \longrightarrow 100_L$
2	141	STR.A	0 1 0 176	$(A) \longrightarrow 176$
2	142	ENT.A	0 0 0 0002	$2 \longrightarrow A$
2	143	STR.A	0 3 0 206	$(A) \longrightarrow 206$
2	144	ENT.A	0 3 0 202	$(202) \longrightarrow A$
2	145	STR.A	0 3 0 204	$(A) \longrightarrow 204$
3	146	ENT.B^n	2 3 0 206	$(206) \longrightarrow B^2$
3	147	BSK.B^n	3 1 0 0000	$(B^3) + 1 \longrightarrow B^3$
2	150	ENT.A	0 1 0 0000	$0 \longrightarrow A$
2	151	STR.A	0 3 3 (3000)	$(A) \longrightarrow 3000 + B^3$
2	152	ENT.A	0 3 0 201	$(201) \longrightarrow A$

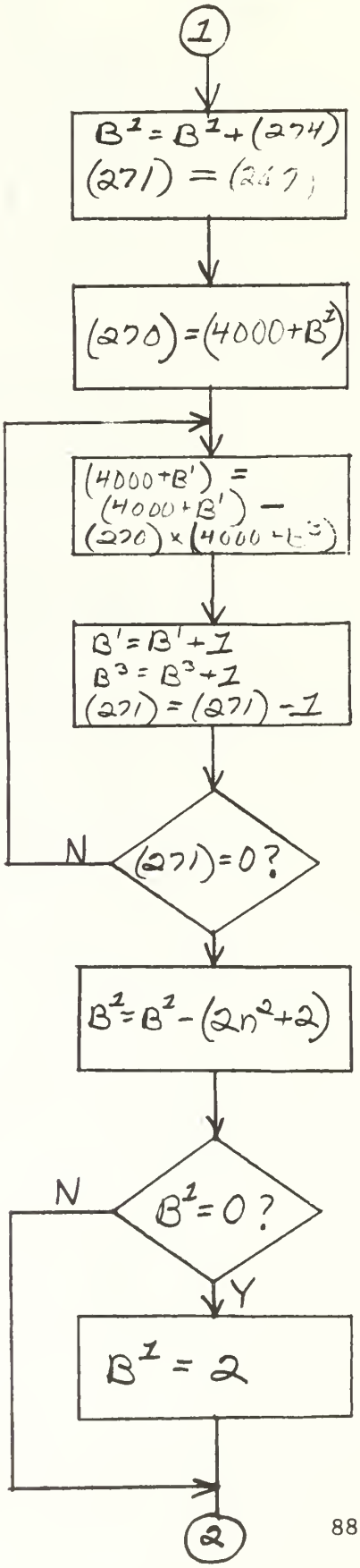
Oper Time	Cell #	Contents (Mnemonic)				Contents (Symbolic)
2	153	STR.A	0	3 0	203	(A) $\longrightarrow$ 203
3	154	ENT.B^n	1	3 0	205	(205) $\longrightarrow$ B^1
2	155	ENT.Q	0	3 1	(1000)	(1000 + B^1) $\longrightarrow$ Q
10	156	MUL	0	3 2	(2000)	(2000 + B^2) x (Q) $\longrightarrow$ AQ
-	157	RSH.AQ	0	1 0	ǯ	Right Shift AQ "ǯ" bits
3	160	RPL.Y + Q	0	3 3	(3000)	(Q) + (3000 + B^3) $\longrightarrow$ 3000 + B^3
3	161	BSK.B^n	1	1 0	0000	(B^1) + 1 $\longrightarrow$ B^1
3	162	STR.B^n	2	7 0	0000	(B^2) $\longrightarrow$ A
2	163	ADD.A	0	3 0	202	(A) + (202) $\longrightarrow$ A
3	164	ENT.B^n	2	7 0	0000	(A) $\longrightarrow$ B^2
3	165	RPL.Y-1	0	3 0	203	(203) - 1 $\longrightarrow$ 203
2	166	JP	5	0 0	155	Jump to 155 if (A) non zero
3	167	RPL.Y + 1	0	3 0	206	(206) + 1 $\longrightarrow$ 206
3	170	RPL.Y-1	0	3 0	204	(204) - 1 $\longrightarrow$ 204
2	171	JP	5	0 0	146	Jump to 146 if (A) non zero
2	172	ENT.A	0	3 0	201	(201) $\longrightarrow$ A
3	173	RPL.A + Y	0	3 0	205	(205) + (A) $\longrightarrow$ 205
3	174	RPL.Y-1	0	3 0	177	(177) - 1 $\longrightarrow$ 177
2	175	JP	5	0 0	142	Jump to 142 if (A) non zero
2	176	JP	1	0 0	(603)	Jump to 603

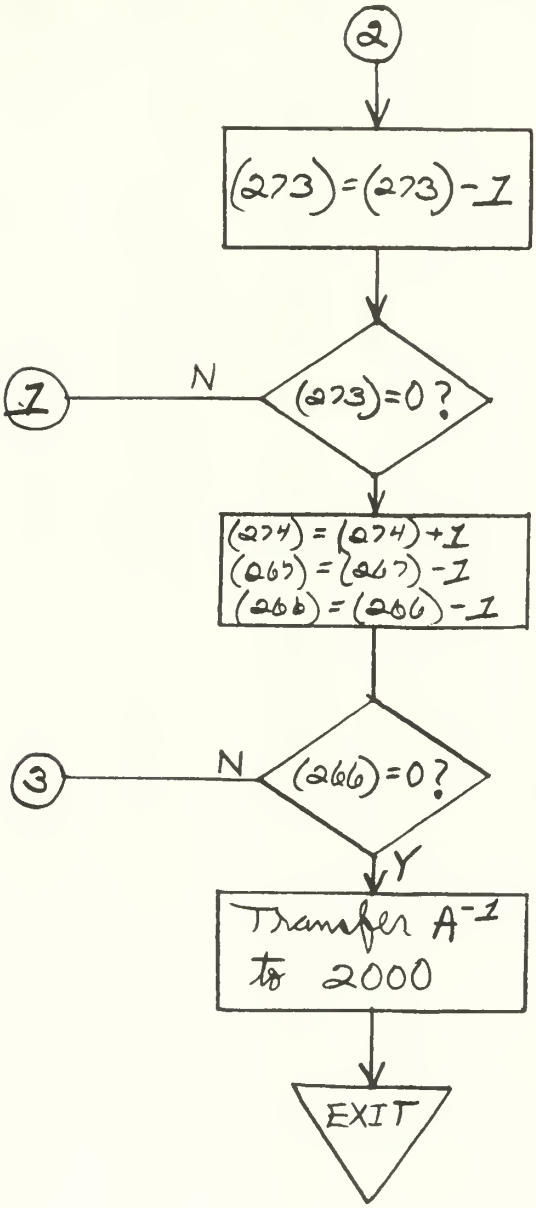
177
201
202
203
204
205
206

Temporary storage and Counters

MATIN Subroutine




1
B² = B² + (274)
(271) = (271)
(270) = (4000 + B²)
(4000 + B') = (4000 + B') − (270) × (4000 + B³)
B' = B' + I
B³ = B³ + I
(271) = (271) − I
N
(271) = 0 ?
B² = B² − (2n² + 2)
N
B² = 0 ?
Y
B² = 2
2


2
(273) = (273) - I
1
N
(273) = 0 ?
(274) = (274) + I
(267) = (267) - I
(266) = (266) - I
3
N
(266) = 0 ?
Y
Transfer A-1
to 2000
EXIT

MATIN Subroutine

Oper Time	Cell #	Contents (Mnemonic)				Contents (Symbolic)
3	600	RTJ	0	0	100	"601" 100_L, "101" $\longrightarrow$ P
-	601	1	0	0	2000	A = 1000, A^{-1} = 2000
-	100	0	0	0	600	
3	101	RPL.Y + 1	0	3	100	(100) + 1 $\longrightarrow$ 100
2	102	STR.A	0	3	106	(A) $\longrightarrow$ 106
2	103	STR.A	0	3	111	(A) $\longrightarrow$ 111
3	104	RPL.Y + 1	0	3	100	(100) + 1 $\longrightarrow$ 100
2	105	STR.A	0	3	265	(A) $\longrightarrow$ 265
2	106	ENT.A	0	2	(601)	(601_U) $\longrightarrow$ A
2	107	STR.A	0	1	117	(A) $\longrightarrow$ 117
2	110	STR.A	0	1	145	(A) $\longrightarrow$ 145
2	111	ENT.A	0	1	(601)	(601_L) $\longrightarrow$ A
2	112	STR.A	0	1	120	(A) $\longrightarrow$ 120
2	113	STR.A	0	1	122	(A) $\longrightarrow$ 122
2	114	STR.A	0	1	256	(A) $\longrightarrow$ 256
3	115	ENT.B^n	1	0	0002	2 $\longrightarrow$ B^1
3	116	ENT.B^n	2	0	0000	0 $\longrightarrow$ B^2
2	117	ENT.A	0	3	(1000)	(1000) $\longrightarrow$ A
2	120	STR.A	0	3	(2000)	(A) $\longrightarrow$ 2000
3	121	BSK.B^n	3	0	0005	(B^2) + 1 $\longrightarrow$ B^2
2	122	STR.A	0	3	(2000)	(A) $\longrightarrow$ 2001
2	123	STR.A	0	3	266	(A) $\longrightarrow$ 266
2	124	STR.A	0	3	272	(A) $\longrightarrow$ 272

Oper Time	Cell #	Contents (Mnemonic)			Contents (Symbolic)
2	125	ADD.A	0 3 0	266	(A) + (266)⟶A
2	126	STR.A	0 3 0	267	(A)⟶267
2	127	STR.A	0 3 0	277	(A)⟶277
3	130	RPL.Y-1	0 3 0	272	(272) - 1⟶272
2	131	ENT.A	0 0 0	0000	0⟶A
2	132	STR.A	0 3 0	274	(A)⟶274
2	133	ENT.Q	0 3 0	266	(266)⟶Q
10	134	MUL	0 3 0	266	(Q) x (266)⟶AQ
10	135	MUL	0 0 0	0002	(Q) x (2)⟶AQ
2	136	ADD.Q	0 0 0	0002	(Q) + 2⟶Q
2	137	STR.Q	0 3 0	276	(Q)⟶276
3	140	ENT.B^n	4 0 0	0002	2⟶B^4
2	141	ENT.A	0 3 0	266	(266)⟶A
2	142	STR.A	0 3 0	271	(A)⟶271
2	143	ENT.A	0 3 0	266	(266)⟶A
2	144	STR.A	0 3 0	270	(A)⟶270
2	145	ENT.A	0 3 1	(1000)	(1000 + B^1)⟶A
2	146	STR.A	0 3 4	4000	(A)⟶4000 + B^4
3	147	BSK.B^n	1 0 0	0000	(B^1) + 1⟶B^1
3	150	BSK.B^n	4 0 0	0000	(B^4) + 1⟶B^4
3	151	RPL.Y-1	0 3 0	270	(270) - 1⟶270
2	152	JP	5 0̄ 0	145	A = 0 : 145⟶P
2	153	ENT.A	0 3 0	266	(266)⟶A
2	154	STR.A	0 3 0	270	(A)⟶270
2	155	ENT.A	0 3 0	270	(270)⟶A

Oper Time	Cell #	Contents (Mnemonic)				Contents (Symbolic)
2	156	SUB.A	5	3 0	271	(A)-(271)⟶A, Skip NI if A ≠ 0
2	157	ENT.Q	1	0 0	0001	1⟶Q, Skip NI
2	160	ENT.Q	0	0 0	0000	0⟶Q
2	161	STR.Q	0	3 4	4000	(Q)⟶4000
3	162	BSK.B^n	4	0 0	0000	$(B^4) + 1 \longrightarrow B^4$
3	163	RPL.Y-1	0	3 0	270	(270) - 1⟶270
3	164	JP	5	0 0	155	A ≠ 0 : 155⟶P
3	165	RPL.Y-1	0	3 0	271	(271) - 1⟶271
2	166	JP	5	0 0	143	A ≠ 0 : 143⟶P
2	167	ENT.Q	0	3 0	277	(277)⟶Q
2	170	ADD.Q	0	0 0	0001	(Q) + 1⟶Q
10	171	MUL	0	3 0	274	(Q) x (274)⟶AQ
-	172	RSH.AQ	0	0 0		Right Shift AQ
2	173	ADD.Q	0	0 0	0000	(Q) + 2⟶Q
3	174	ENT.B^n	1	0 0	0000	$0 \longrightarrow B^1$
2	175	STR.Q	0	3 0	275	(Q)⟶275
2	176	ENT.A	0	3 0	267	(267)⟶A
2	177	STR.A	0	3 0	271	(A)⟶271
2	200	ENT.A	0	3 0	272	(272)⟶A
2	201	STR.A	0	3 0	273	(A)⟶273
2	202	ENT.A	0	3 1	4000	$(4000 + B^1) \longrightarrow A$
2	203	STR.A	0	3 0	270	(A)⟶270
2	204	ENT.A	0	0 0	0000	0⟶A
2	205	ENT.Q	0	3 1	4000	$(4000 + B^1) \longrightarrow Q$
12	206	DIV	0	3 0	270	AQ + (270)⟶Q

Oper Time	Cell #	Contents (Mnemonic)				Contents (Symbolic)
2	207	STR.Q	0 3 1	4000		(Q)⟶4000 + B^1
3	210	BSK.B^n	1 0 0	0000		(B^1) + 1⟶B^1
3	211	RPL.Y-1	0 3 0	271		(271) - 1⟶271
2	212	JP	5 0 0	204		A ≠ 0 : 204⟶P
3	213	STR.B^n	1 4 0	0000		(B^1)⟶A
2	214	ADD.A	0 3 0	274		(A) + (274)⟶A
3	215	ENT.B^n	1 7 0	0000		(A)⟶B^1
2	216	ENT.A	0 3 1	4000		(4000 + B^1)⟶A
2	217	STR.A	0 3 0	270		(A)⟶270
2	220	ENT.A	0 3 0	267		(267)⟶A
2	221	STR.A	0 3 0	271		(A)⟶271
3	222	ENT.B^n	3 3 0	275		(275)⟶B^3
2	223	ENT.Q	0 3 3	4000		(4000 + B^3)⟶Q
10	224	MUL	0 3 0	270		(Q) x (270)⟶AQ
-	225	RSH.AQ	0 0 0	-		Right Shift AQ
3	226	RPL.Y-Q	0 3 1	4000		(4000 + B^1) - (Q)⟶(4000+B^1)
3	227	BSK.B^n	1 0 0	0000		(B^1) + 1⟶B^1
3	230	BSK.B^n	3 0 0	0000		(B^3) + 1⟶B^3
3	231	RPL.Y-1	0 3 0	271		(271) - 1⟶271
2	232	JP	5 0 0	223		A ≠ 0 : 223⟶P
3	233	STR.B^n	1 4 0	0000		(B^1)⟶A
2	234	SUB.A	5 3 0	276		(A)-(276)⟶A, Skip NI if A≠0
3	235	ENT.B^n	1 0 0	0002		2⟶B^1
3	236	RPL.Y-1	0 3 0	273		(273) - 1⟶273
2	237	JP	5 0 0	213		A ≠ 0 : 213⟶P
3	240	RPL.Y+1	0 3 0	274		(274) + 1⟶274

Oper Time	Cell #	Contents (Mnemonic)		Contents (Symbolic)
3	241	RPL.Y-1	0 3 0 267	(267) - 1 $\longrightarrow$ 267
3	242	RPL.Y-1	0 3 0 266	(266) - 1 $\longrightarrow$ 266
2	243	JP	5 0 0 167	A $\neq$ 0 : 167 $\longrightarrow$ P
3	244	BSK.B^n	2 0 0 0000	(B^2) + 1 $\longrightarrow$ B^2
3	245	RPL.Y-1	0 3 0 272	(272) + 1 $\longrightarrow$ 272
2	246	STR.A	0 3 0 266	(A) $\longrightarrow$ 266
3	247	ENT.B^n	4 0 0 0002	2 $\longrightarrow$ B^4
3	250	STR.B^n	4 4 0 0000	(B^4) $\longrightarrow$ A
2	251	ADD.A	0 3 0 266	(A) + (266) $\longrightarrow$ A
3	252	ENT.B^n	4 7 0 0000	(A) $\longrightarrow$ B^4
2	253	ENT.A	0 3 0 266	(266) $\longrightarrow$ A
2	254	STR.A	0 3 0 273	(A) $\longrightarrow$ 273
2	255	ENT.A	0 3 4 4000	(4000 + B^4) $\longrightarrow$ A
2	256	STR.A	0 3 2 (2000)	(A) $\longrightarrow$ 2000 + B^2
3	257	BSK.B^n	4 0 0 0000	(B^4) + 1 $\longrightarrow$ B^4
3	260	BSK.B^n	2 0 0 0000	(B^2) + 1 $\longrightarrow$ B^2
3	261	RPL.Y-1	0 3 0 273	(273) - 1 $\longrightarrow$ 273
2	262	JP	5 0 0 255	A $\neq$ 0 : 255 $\longrightarrow$ P
3	263	RPL.Y-1	0 3 0 272	(272) - 1 $\longrightarrow$ 272
2	264	JP	5 0 0 250	A $\neq$ 0 : 250 $\longrightarrow$ P
2	265	JP	1 0 0 (602)	602 $\longrightarrow$ P
	266 267 270 271 272 273 274 275 276 277	. Temporary storage and Counters		

APPENDIX VI

TECHNICAL CHARACTERISTICS OF UNIVAC 1830 COMPUTER

This Appendix contains a brief listing of the technical characteristics taken from [1], a description of the instruction word format, and a listing of the instructions used with the 1830 computer.

Technical Characteristics

Memory

4096 words, 30 bits

4 microsecond cycle time

Control

Single Address

62 instructions

Seven index register

Seven branch designators

Seven operand interpretation designators

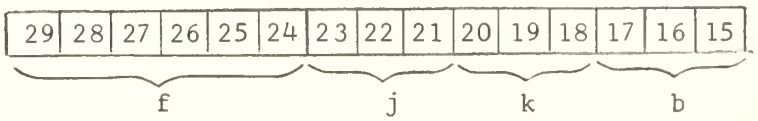
Instruction Execution Time

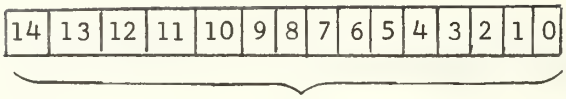
Add:	8 microseconds
Subtract:	8 microseconds
Multiply:	32-48 microseconds
Divide:	48 microseconds
Jump:	8 microseconds

Construction

Semiconductor Microelectronic Integrated Circuits

Instruction Word Format





f - Function Code Designator

j - Branch Condition Designator

k - Operand Interpretation Designator

b - Index Designator

y - Operand Designator

NTDS UNIT COMPUTER

AN/USQ-20

Repertoire of Instructions

01	Right SHift • Q	Shift (Q) Right by Y
02	Right SHift • A	Shift (A) Right by Y
03	Right SHift • AQ	Shift (AQ) Right by Y
04*	COMpare • A, • Q, • AQ	Sense (j); $(A)_i = (A)_f$
05	Left SHift • Q	Shift (Q) Left by Y
06	Left SHift • A	Shift (A) Left by Y
07	Left SHift • AQ	Shift (AQ) Left by Y
10	ENTer • Q	Y → Q
11	ENTer • A	Y → A
12	ENTer • B^n	Y → B^j
13^	EXternal - FunCTion • C^n	$\hat{j} \neq 0$ or 1, (Y) → $C^{\hat{j}}$, $\hat{j}$=0 or 1, See Note
14	STоRe • Q	(Q) → Y; k=0, Q' → Q
15	STоRe • A	(A) → Y; k=4, A' → A
16	STоRe • B^n	$(B)^j$ → Y
17^	STоRe • C^n	$(C)^{\hat{j}}$ → Y
20	ADD • A	(A) + Y → A
21	SUBtract • A	(A) − Y → A
22	MULtiply	(Q) Y → AQ
23*	DIVide	(AQ) / Y → Q. R → A_f
24	RePLace • A+Y	(A) + (Y) → Y & A
25	RePLace • A−Y	(A) − (Y) → Y & A
26*	ADD • Q	(Q) + Y → Q. $(A)_i = (A)_f$ } j interpretation
27*	SUBtract • Q	(Q) − Y → Q. $(A)_i = (A)_f$ } reversed for A&Q
30	ENTer • Y+Q	Y + (Q) → A
31	ENTer • Y−Q	Y − (Q) → A
32	STоRe • A+Q	(A) + (Q) → Y & A
33	STоRe • A−Q	(A) − (Q) → Y & A
34	RePLace • Y+Q	(Y) + (Q) → Y & A
35	RePLace • Y−Q	(Y) − (Q) → Y & A
36	RePLace • Y+1	(Y) + 1 → Y & A
37	RePLace • Y−1	(Y) − 1 → Y & A
40*	ENTer • LP**	L[Y(Q)] → A; j=2, even parity, j=3, odd parity
41	ADD • LP	L[Y(Q)] + (A) → A
42	SUBtract • LP	(A) − L[Y(Q)] → A
43	COMpare • MASK	(A) − L[Y(Q)] SENSE (j), (A) + L[Y(Q)]; $(A)_j = (A)_f$
44*	RePLace • LP	L(Y)(Q) → Y & A; j=2, even parity; j=3, odd parity
45	RePLace • A+LP	L(Y)(Q) + (A) → Y & A
46	RePLace • A−LP	(A) − L(Y)(Q) → Y & A
47	STоRe • LP	L(A)(Q) → Y; $(A)_i = (A)_f$
50	SELective • SET	SET $(A)_n$ FOR $Y_n = 1$
51	SELective • CP**	COMPLEMENT $(A)_n$ FOR $Y_n = 1$
52	SELective • CL**	CLEAR $(A)_n$ FOR $Y_n = 1$
53	SELective • SU**	Y_n → $(A)_n$ FOR $(Q)_n = 1$
54	Replace SElective • SET	SET $(A)_n$ FOR $(Y)_n = 1$, → Y & A
55	Replace SElective • CP	COMPLEMENT $(A)_n$ FOR $(Y)_n = 1$, → Y & A
56	Replace SElective • CL	CLEAR $(A)_n$ FOR $(Y)_n = 1$, → Y & A
57	Replace SElective • SU	$(Y)_n$ → $(A)_n$ FOR $(Q)_n = 1$, → Y
60	JumP (arithmetic)	} Jump to Y if j-condition is satisfied.
61	JumP (manual)	} (see JP & RJP j - Designators)
62^	JumP (if • C^n has ACTIVE INput buffer)	Jump to Y if $C^{\hat{j}}$ input buffer active } (see JP & RJP j - Designators)
63^	JumP (if • C^n has ACTIVE OUTput buffer)	Jump to Y if $C^{\hat{j}}$ output buffer active } (see JP & RJP j - Designators)
64	Return JumP (arithmetic)	} Jump to Y + 1 and P + 1 → Y_L if j condition is
65	Return JumP (manual)	} satisfied (see JP & RJP j - Designators)
66^	TERMinate • C^n • INPUT	Terminate input buffer on channel $\hat{j}$
67^	TERMinate • C^n • OUTPUT	Terminate output buffer on channel $\hat{j}$
70*	RePeoT	Execute NI Y times
71	BSKip • B^n	$(B)^j$ = Y, skip NI and clear $(B)^j$, $(B)^j \neq$ Y, Advance B^j and read NI
72	BJumP • B^n	$(B)^j = 0$, read NI; $(B)^j \neq 0$, $(B)^j - 1$ and jump to address Y
73^	INput • C^n (without monitor mode)	Buffer IN on $C^{\hat{j}}$; $\hat{k}=3$, (Y) → $(00100+\hat{j})$; $\hat{k}=1$, $(Y)_L$ → $(00100+\hat{j})_L$; $\hat{k}=0$, Y → $(00100+\hat{j})_L$.
74^	OUTput • C^n (without monitor mode)	Buffer OUT on $C^{\hat{j}}$, $\hat{k}=3$, (Y) → $(00120+\hat{j})$; $\hat{k}=1$, $(Y)_L$ → $(00120+\hat{j})_L$; $\hat{k}=0$, Y → $(00120+\hat{j})_L$.
75^	INput • C^n (with • MONITOR mode)	Buffer IN on $C^{\hat{j}}$ with mon. $\hat{k}=3$, (Y) → $(00100+\hat{j})$; $\hat{k}=1$, $(Y)_L$ → $(00100+\hat{j})_L$; $\hat{k}=0$, Y → $(00100+\hat{j})_L$. mon. inter. at $00040+\hat{j}$
76^	OUTput • C^n (with • MONITOR mode)	Buffer OUT on $C^{\hat{j}}$ with mon. $\hat{k}=3$, (Y) → $(00120+\hat{j})$; $\hat{k}=1$, $(Y)_L$ → $(00120+\hat{j})_L$; $\hat{k}=0$, Y → $(00120+\hat{j})_L$. mon. inter. at $00060+\hat{j}$

— NO - OPerotion	} CS-1 Mono - codes
— ComPlement • A or • Q	
— CLear • A, • Q, • B^n, or Y	
— Remove Interrupt Lockout	
— Remove Interrupt Lockout and JumP • Y	
— TEST • CO or • CI	

**LP - Logical Product CP - Complement SU - Substitute CL - Clear *^ Special j & k Designators (see opposite side of card) Y - The operand, Y or (Y)

NOTE: Skip NI if other Computer (on channel 0 or 1) has input buffer active. Execute twice.

NTDS UNIT COMPUTER

AN/USQ-20 *Repertoire of Instructions*

j-DESIG.

j	(Not applicable on * or ^) Skip Code
0	(no skip)
1	SKIP
2	Q POS
3	Q NEG
4	A ZERO
5	A NOT Zero
6	A POS
7	A NEG

NORMAL k-DESIGNATORS

k	READ Code	READ Origin	STORE Code	STORE Dest.	REPLACE Code	REPLACE Origin	REPLACE Dest
0	'blank'	U_L	Q	Q	'not used'	—	—
1	L	M_L	L	M_L	L	M_L	M_L
2	U	M_U	U	M_U	U	M_U	M_U
3	W	M	W	M	W	M	M
4	X	XU_L	A	A	'not used'	—	—
5	LX	XM_L	CPL	Cpl M_L	LX	XM_L	M_L
6	UX	XM_U	CPU	Cpl M_U	UX	XM_U	M_U
7	A	A	CPW	Cpl M	'not used'	—	—

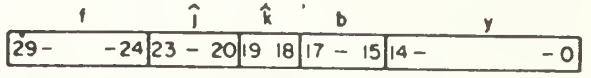
LEGEND

M – Memory word (30 bits) Cpl – Complement
M_L – Lower half memory word A – A-register
M_U – Upper half memory word Q – Q-register
X – Sign bit extended U – U-register

$\hat{j}$-DESIGNATORS

(4 bits)

$\hat{j}$ Occupies 4 bit positions and represents C^n where n may be $0-15_8$
The instruction word assumes the format.



$\hat{k}$-DESIGNATORS

(2 bits)

k	EX–FCT 13	STR•C^n 17	JP 62 63	IN•C^n, OUT•C^n 73 75 74 76
0	'not used'	'not used'	'blank'	'blank'
1	'not used'	'not used'	L	L
2	'not used'	'not used'	U	'not used'
3	W	W	W	W

JP & RJP j-DESIGNATORS

j	JP 60	RJP 64	JP 61	RJP 65
0	(No Jump)*		(Uncond Jump)	
1	(Uncond. Jump)*		KEY 1	
2	Q POS		KEY 2	
3	Q NEG		KEY 3	
4	A ZERO		STOP	
5	A NOT Zero		STOP 5	
6	A POS		STOP 6	
7	A NEG		STOP 7	
$\hat{j}$	62 $\hat{j}$		63 $\hat{j}$	
$0-15_8$	C^n ACTIVE IN		C^n ACTIVE OUT	

*60 Clears interrupt & bootstrap modes.

*j-DESIGNATORS

j	COM•A, •Q, •AQ 04	DIV 23	ADD•Q, SUB•Q 26 27	ENT•LP, RPL•LP 40 44	RPT 70	
0	(no skip)	(no skip)	(no skip)	(no skip)	(no mod.)	Y of NE = Y
1	(unconditionol skip)	SKIP	SKIP	SKIP	ADV	Y of NE = Y+1
2	Y LESS Y ≤ (Q)	NO Over Flow	A POS	EVEN parity	BACK	Y of NE = Y–1
3	Y MORE Y > (Q)	Over Flow	A NEG	ODD parity	ADD B	Y of NE = Y + B^b
4	Y IN (Q) ≥ Y and Y > (A)	A ZERO	Q ZERO	A ZERO	Rpl. Inc.	Y of NE = Y [+ B^6] ✓
5	Y OUT (Q) < Y or Y ≤ (A)	A NOT Zero	Q NOT Zero	A NOT Zero	ADV R	Y of NE = Y+1 [+ B^6] ✓
6	Y LESS Y ≤ (A)	A POS	Q POS	A POS	BACK R	Y of NE = Y–1 [+ B^6] ✓
7	Y MORE Y > (A)	A NEG	Q NEG	A NEG	ADD B R	Y of NE = Y + B^b [+ B^6] ✓

✓ B^6 Increment if NI is RPL class; increments Y address for the store portion of the replace.
NE – Next execution

APPENDIX VII

KALMAN FILTER EQUATIONS

This Appendix contains a listing of the Kalman Filter equations, the definitions of the terms, the equivalent terms used in the test program, and the dimensions used to compute the program running time.

Equations

$$\underline{\overset{*}{X}}(k/k) = \underline{X}(k/k-1) + G(k)\left[\underline{Z}(k) - \underline{\hat{Z}}(k/k-1)\right]$$

$$G(k) = \left[P(k/k-1)H^T(HP(k/k-1)H^T + R)^{-1}\right]$$

$$P(k/k-1) = \underline{\underline{\Phi}}\left[P(k-1/k-2) - G(k-1)HP(k-1/k-2)\right]\underline{\underline{\Phi}}^T + Q$$

$$\underline{Z}(k/k-1) = H\underline{\hat{X}}(k/k-1) = H\underline{\underline{\Phi}}\,\underline{\overset{*}{X}}(k-1/k-1)$$

Definition of Terms

Variable	Definition	Remarks
$\underline{\underline{\Phi}}$	nxn state transition matrix	constant for a given time-invariant plant
G	nxl optimum filter gain matrix	
H	plant observability matrix	order of matrix depends upon number of plant observables
R	measurement noise covariance matrix	value selected by operator
Q	random excitation distribution covariance matrix	value selected by operator
P	error covariance matrix	initial value selected by operator
$\underline{Z}$	nxl vector of noisy plant observable states	

Variable	Definition	Remarks
$\hat{\underline{X}}$	nxl vector of the predicted value of $\underline{X}(k)$, based on the previous observation $\underline{X}(k - 1)$	
$\hat{\underline{Z}}$	nxl vector of the predicted value of $\underline{Z}(k)$, based on the previous observation $\underline{Z}(k - 1)$	
$\overset{*}{\underline{X}}$	nxl vector of the best estimate of $\underline{X}(k)$, based on the current observation of $\underline{Z}(k)$	

Equivalence and Dimensions of Terms

Variable	Programming Equivalent	Dimensions
$\underline{\underline{\Phi}}$	PHI	4 x 4
G	G	4 x 1
H	H	1 x 4
R	R	1 x 1
Q	Q	4 x 4
P	P	4 x 4
$\underline{Z}$	Z	1 x 1
$\hat{\underline{X}}$	XH	4 x 1
$\hat{\underline{Z}}$	ZH	1 x 1
$\overset{*}{\underline{X}}$	XS	4 x 1
$\underline{\underline{\Phi}}^T$	PHIT	4 x 4
H^T	HT	4 x 1

APPENDIX VIII

KALMAN FILTER PROGRAMS

This Appendix contains a listing of the Kalman Filter programs for both computers. These programs are a straight-forward implementation of the equations shown in Appendix VII and in fact appear very similar. The operation times shown are for the complete subroutine or microprogram "called", and are based on the matrix dimensions listed in the previous Appendix.

Here again, subroutine names and variables are used to denote the starting addresses in memory of the appropriate programs or matrices. The terms TA, TB, and TC refer to sections of memory used as temporary storage.

All constant matrices and the necessary transposes are considered to be resident in memory and the input matrix, Z, is also considered to be present when needed since I/O programming was not considered.

KALMAN FILTER PROGRAM

for the Microprogrammed Computer

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
2093	00	MATML, G	G x H ⟶TC
	01	H , TC	
9005	02	MATML, TC	TC x P ⟶TA
	03	P , TA	
680	04	MATSB, P	P-TA ⟶TC
	05	TA , TC	
9005	06	MATML, PHI	PHI x TC ⟶TA
	07	TC , TA	
9005	10	MATML, TA	TA x PHIT⟶TC
	11	PHIT , TC	
712	12	MATAD, TC	TC + Q⟶P
	13	Q , P	
2321	14	MATML, H	H x P⟶TA
	15	P , TA	
788	16	MATML, TA	TA x HT⟶TC
	17	HT , TC	
179	20	MATAD, TC	TC + R⟶TA
	21	R , TA	
343	22	MATIN, TA	$(TA)^{-1}$⟶TB
	23	TB	
2021	24	MATML, P	P x HT⟶TC
	25	HT , TC	

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
764	26	MATML, TC	TC x TB ⟶ G
	27	TB , G	
2021	30	MATML, PHI	PHI x XS ⟶ TA
	31	XS , TA	
788	32	MATML, H	H x TA ⟶ TB
	33	TA , TB	
177	34	MATSB, Z	Z - TB ⟶ TC
	35	TB , TC	
767	36	MATML, G	G x TC ⟶ TB
	37	TC , TB	
308	40	MATAD, TA	TA + TB ⟶ XS
	41	TB , XS	

KALMAN FILTER PROGRAM

for the UNIVAC 1830 Computer

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
1046	00	RTJ 000, MATML	G x H⟶TC
	01	G , H	
	02	0 , TC	
2534	03	RTJ 000, MATML	TC x P⟶TA
	04	TC , P	
	05	0 , TA	
257	06	RTJ 000, MATSB	P-TA⟶TC
	07	P , TA	
	10	-1 , TC	
2534	11	RTJ 000,MATML	PHI x TC⟶TA
	12	PHI , TC	
	13	0 , TA	
2534	14	RTJ 000, MATML	TA x PHIT⟶TC
	15	TA , PHIT	
	16	0 , TC	
257	17	RTJ 000, MATML	TC + Q⟶P
	20	TC , Q	
	21	0 , P	
692	22	RTJ 000, MATML	H x P⟶TA
	23	H , P	
	24	0 , TA	
245	25	RTJ 000, MATML	TA x HT⟶TC

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
	26	TA , HT	
	27	0 , TC	
92	30	RTJ 000, MATAD	TC + R $\longrightarrow$ TA
	31	TC , R	
	32	+1 , TA	
249	33	RTJ 000, MATIN	$(TA)^{-1} \longrightarrow TB$
	34	TA , TB	
746	35	RTJ 000, MATML	P x HT $\longrightarrow$ TC
	36	P , HT	
	37	0 , TC	
374	40	RTJ 000, MATML	TC x TB $\longrightarrow$ G
	41	TC , TB	
	42	0 , G	
746	43	RTJ 000, MATML	PHI x XS $\longrightarrow$ TA
	44	PHI , XS	
	45	0 , TA	
245	46	RTJ 000, MATML	H x TA $\longrightarrow$ TB
	47	H , TA	
	50	0 , TB	
92	51	RTJ 000, MATSB	Z-TB $\longrightarrow$ TC
	52	Z , TB	
	53	-1 , TC	
374	54	RTJ 000, MATML	G x TC $\longrightarrow$ TB
	55	G , TC	
	56	0 , TB	

Oper Time	Cell #	Contents (Mnemonic)	Contents (Symbolic)
125	57	RTJ 000, MATAD	TA + TB ⟶ XS
	60	TA , TB	
	61	+1 , XS	

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	20
2.	Library Naval Postgraduate School Monterey, California 93940	2
3.	COMMANDANT (EEE) U.S. Coast Guard 1300 E. St. NW. Washington, D.C. 20226	1
4.	Prof. Mitchell L. Cotton Department of Electrical Engineering Naval Postgraduate School Monterey, California 93940	3
5.	LT Robert A. Ingalls, USCG c/o COMMANDER 1st Coast Guard District 1400 Custom House Boston, Mass. 02109	1

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Postgraduate School Monterey, California 93940	UNCLASSIFIED
	2b. GROUP

3. REPORT TITLE

LOGICAL DESIGN OF A MICROPROGRAMMED SPECIAL-PURPOSE COMPUTER

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Thesis, M.S., December 1966

5. AUTHOR(S) (Last name, first name, initial)

INGALLS, Robert Austin

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1966	109	4

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. AVAILABILITY/LIMITATION NOTICES

[redacted]

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

An investigation is made of the microprogramming approach to logical design of a digital machine. The digital processor designed performs general matrix manipulation and in particular is adapted to the requirements for implementing a Kalman-Weiner filter in a Track-While-Scan radar system.

A timing and data-flow analysis is made and micro-programs written to implement the required macros for the selected benchmark application. A detailed comparison is made of the operation of the microprogrammed processor to that of a current general purpose avionics type computer of similar main memory cycle time. Some conclusions are made concerning the optimizing of various parameters of a microprogrammed computer design.

DD FORM 1 JAN 64 1473

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computer Design Microprogramming Matrix Arithmetic						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2*a*. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking ia to be in accordance with appropriate security regulationa.

2*b*. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital lettera. Titles in all cases ahould be unclaasified. If a meaningful title cannot be selected without claasification, ahow title classification in all capitals in parentheais immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progresa, summary, annual, or final. Give the inclusive dates when a apecific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, uae date of publication.

7*a*. TOTAL NUMBER OF PAGES: The total page count ahould follow normal pagination procedures, i.e., enter the number of pages containing information.

7*b*. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8*a*. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8*b*, 8*c*, & 8*d*. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, aubproject number, ayatem numbera, task number, etc.

9*a*. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. Thia number must be unique to this report.

9*b*. OTHER REPORT NUMBER(S): If the report has been asaigned any other report numbers *(either by the originator or by the sponsor)*, alao enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further disaemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC ia not authorized."

(3) "U. S. Government agenciea may obtain copies of this report directly from DDC. Other qualified DDC users shall request through

_____."

(4) "U. S. military agencies may obtain copiea of this report directly from DDC. Other qualified users ahall request through

_____."

(5) "All distribution of thia report is controlled. Qualified DDC users ahall request through

_____."

If the report haa been furnished to the Office of Technical Services, Department of Commerce, for aale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may alao appear elaewhere in the body of the technical report. If additional space ia required, a continuation sheet ahall be attached.

It is highly desirable that the abatract of claaaified reports be unclaasified. Each paragraph of the abatract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS)*, *(S)*, *(C)*, *or (U)*.

There ia no limitation on the length of the abstract. However, the suggested length ia from 150 to 225 words.

14. KEY WORDS: Key worda are technically meaningful terms or short phrases that characterize a report and may be used aa index entries for cataloging the report. Key words must be aelected so that no aecurity clasaification is required. Identifiers, auch aa equipment model deaignation, trade name, military project code name, geographic location, may be uaed as key worda but will be followed by an indication of technical context. The aasignment of links, rolea, and weighta ia optional.

DD FORM 1 JAN 64 1473 (BACK)